

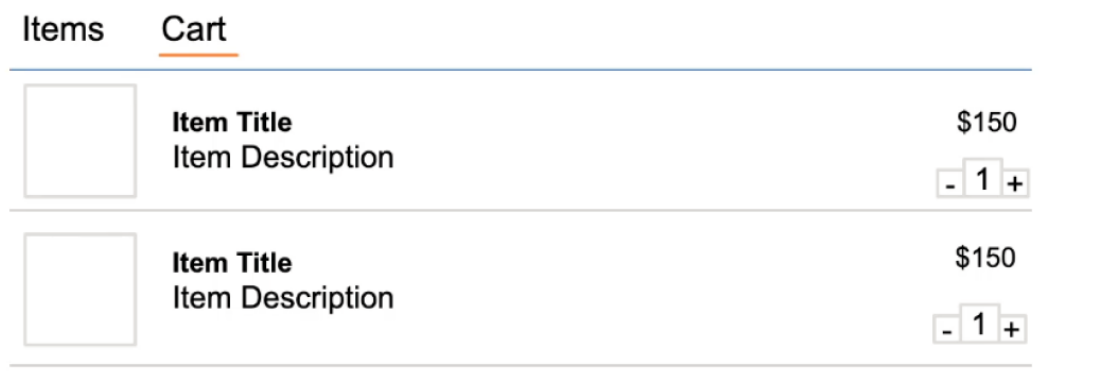
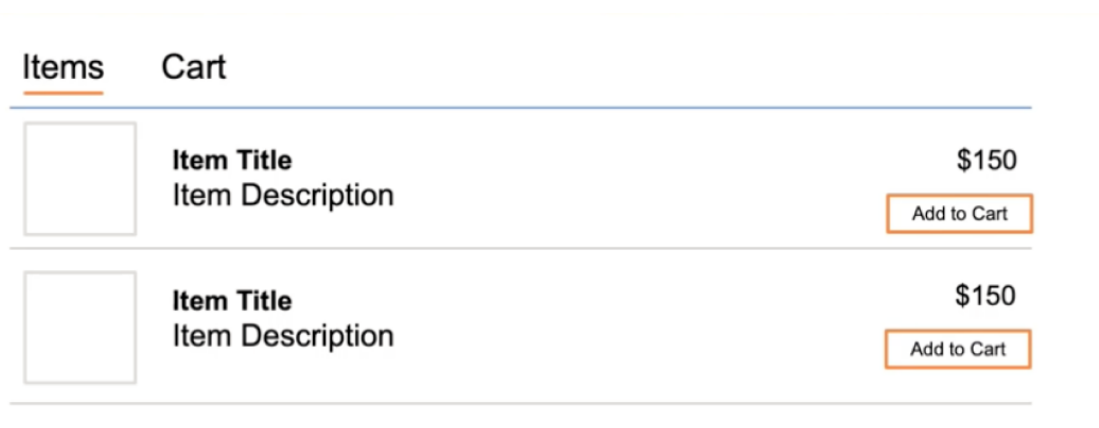
In this lesson we prepare to build more complex React apps by learning to think in terms of React Components. To illustrate thinking in Components we follow a four step process.

Four Step Process

- Make a sketch of the result
- Divide the sketch into Components
- Name each Component
- Write the code

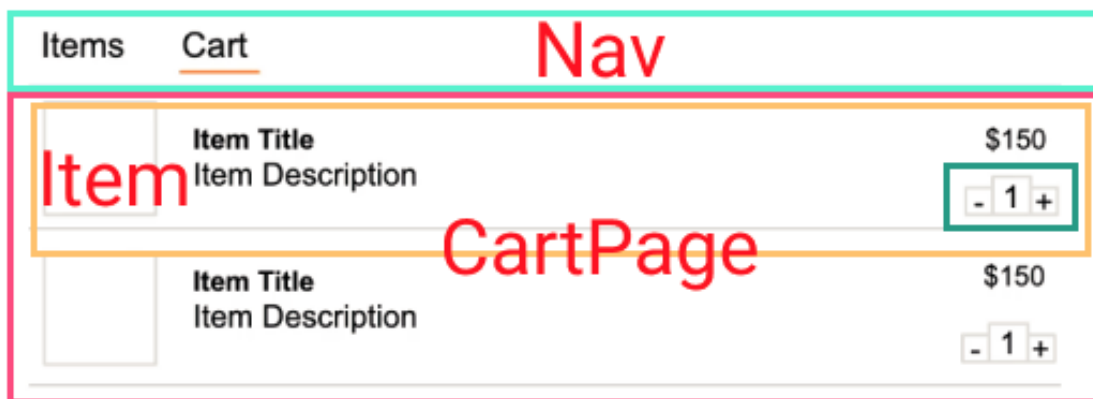
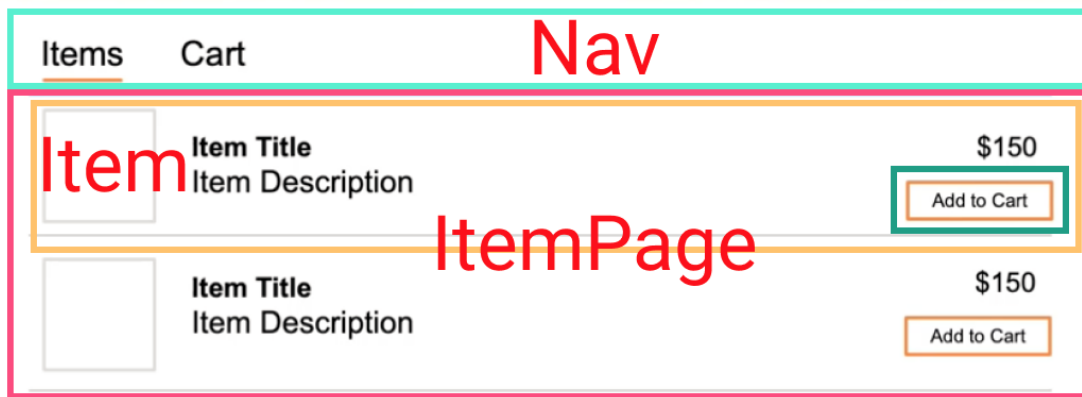
Make a Sketch of the Result

In this course we will build a simple e-commerce site with an **Items page** and a **Cart page**. We start by preparing a **sketch** of what each page will look like while also keeping functionality in mind.



Divide the Sketch into Components

The next step is to **divide** the **sketch** into **Components**. Ask yourself what parts of this page can be chunked up and reassembled keeping in mind reusability and the composable design thinking of React. Notice how the **Nav** and **Item** Components are reusable across both the **ItemPage** and **CartPage** Components and are used within them. Also, notice how the **Item** Component has **almost** the same usage across **ItemPage** and **CartPage**. We reuse this Component by passing it **children** (**Add to Cart** button on **ItemPage** and **Quantity** on **CartPage**).



Name Each Component

Next, attach a name to each Component. In this example we have an **App** Component that handles all the interactions between the main Components. We can choose names for these Components such as **Nav**, **ItemPage**, **CartPage**, and **Item**. Notice in the above images that we can label our chunked up sketch for clarity before we start coding.

Write the Code

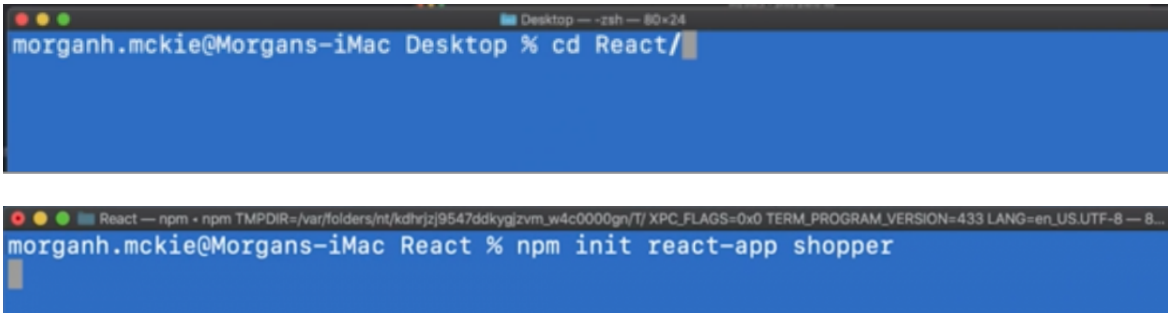
With a high level design for our app we can start coding. Before we create the React **Components**, we will once again use **Create React App** to scaffold a base React application for us. Using your **terminal**, return to the location where you chose to store your code files. In this lesson example, we return to our **React** folder created on the **Desktop**. From the terminal **run** the following **Create React App** command from **NPM** to setup our initial **shopper** project.

```
npm init react-app shopper
```

```
Desktop -- zsh -- 80x24
Last login: Fri Nov 13 09:33:55 on ttys000
morganh.mckie@Morgans-iMac ~ % cd Desktop
morganh.mckie@Morgans-iMac Desktop %
```

Intro to Frontend Development with React

Think in Components

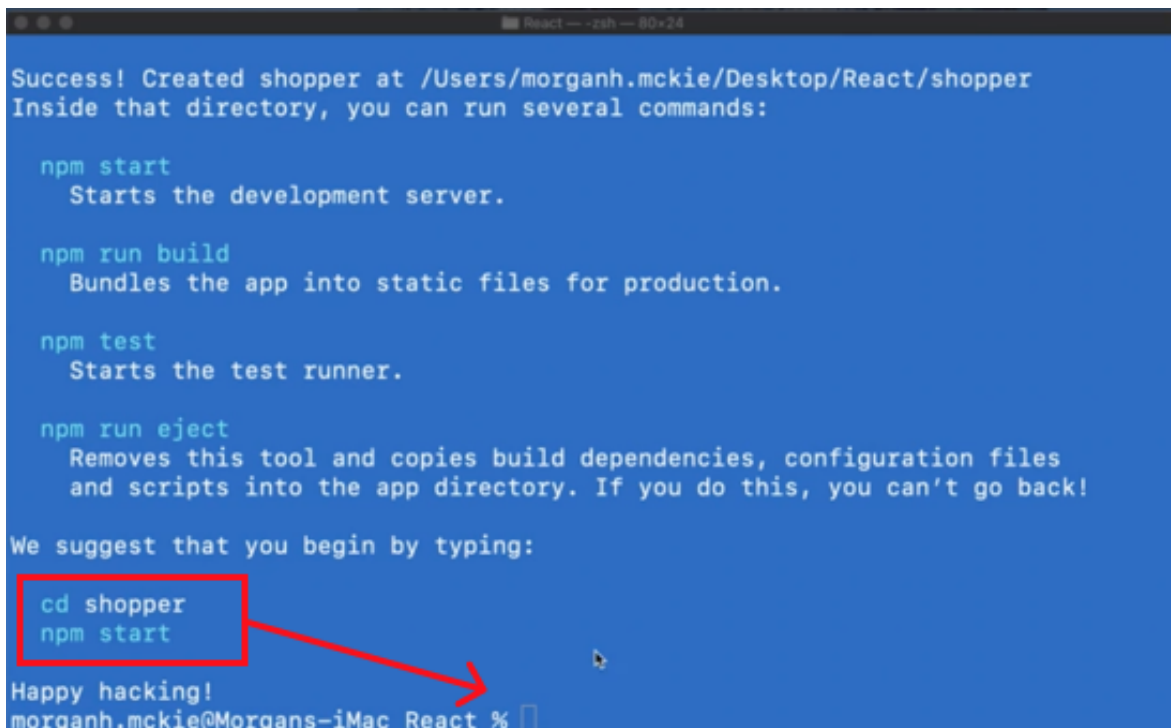


The image shows two terminal windows. The first window shows the user navigating to the 'React' directory. The second window shows the user running 'npm init react-app shopper' to create a new React application named 'shopper'.

```
morganh.mckie@Morgans-iMac Desktop % cd React/

morganh.mckie@Morgans-iMac React % npm init react-app shopper
```

Change into the **folder** just created, **shopper**, in this example and start the new app with **npm start**.



The image shows a terminal window with the output of the 'npm init react-app shopper' command. It lists several commands and their functions: 'npm start' (Starts the development server), 'npm run build' (Bundles the app into static files for production), 'npm test' (Starts the test runner), and 'npm run eject' (Removes this tool and copies build dependencies, configuration files and scripts into the app directory). A red box highlights the commands 'cd shopper' and 'npm start', with a red arrow pointing to the prompt 'morganh.mckie@Morgans-iMac React %'.

```
Success! Created shopper at /Users/morganh.mckie/Desktop/React/shopper
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd shopper
  npm start

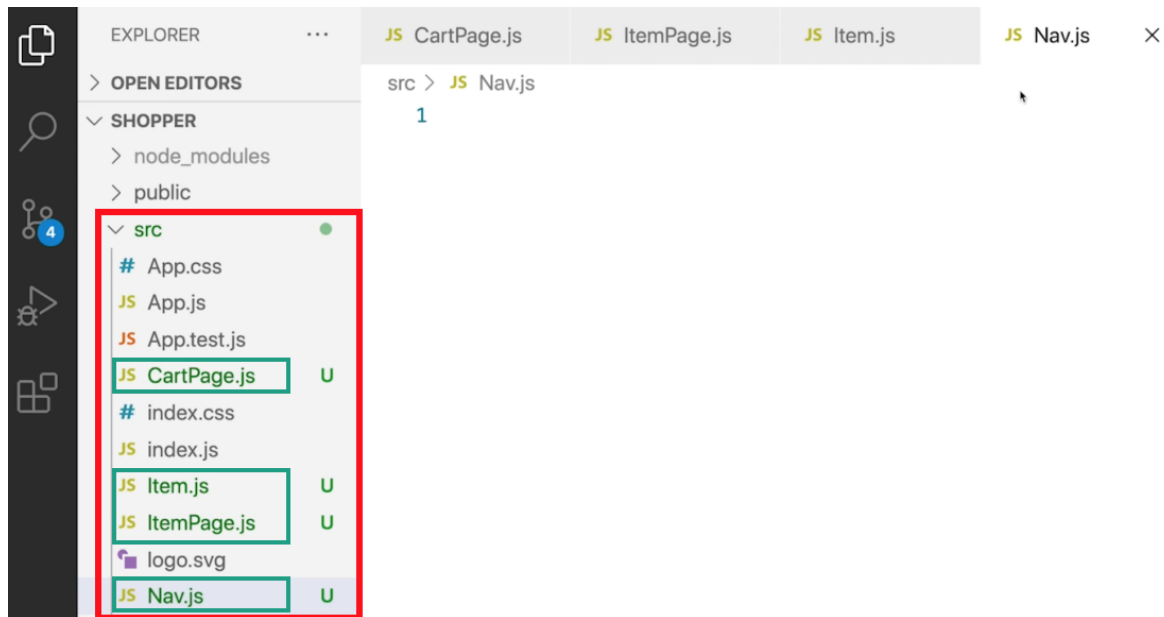
Happy hacking!
morganh.mckie@Morgans-iMac React %
```

```
cd shopper
npm start
```

Inside **VS Code**, choose from the menu **File | Open Folder**. Navigate to the location of the **shopper** folder we just created and **open** the project. Inside the **VS Code File Explorer** create the four **Components** with the **names** we identified above: **Nav**, **ItemPage**, **CartPage**, and **Item**. Notice each Component file name starts with a capital letter and has a JavaScript file extension (**.js**).

Intro to Frontend Development with React

Think in Components



In the next lesson we will start to build our App Components.

In the last lesson we designed our app and divided it up into React Components. We then opened the project in VS Code and added the four components: **ItemPage**, **CartPage**, **Item** and **Nav**. In this lesson we start to code the **App** Component and **Nav** Component.

App Component

In VS Code, open the file **App.js** and delete all of its content (boilerplate code from Create React App). Let's start by importing the **React library** as well as our **App.css** stylesheet file (delete all of its content boilerplate code from Create React App). Notice the **difference** in the **imports**. For installed node modules (from NPM) we import from file name inside quotes, "react". For our own files we use the relative path from this current file to the file we are importing. If the file is in the same folder we use a dot slash (./) as in "./App.css".

```
import React from 'react';
import './App.css';
```

Next, we create our App Component. There are three widely used methods for creating React Components. Using JavaScript's **Class** syntax, **Function Declaration** syntax, or **Arrow Function** syntax.

```
import React from 'react';
import './App.css';
```

```
function App() {
```

```
}
```

```
export default App;
```

The code above is how we would set up our App Component using a regular Function Declaration (as we did in our **HelloWorld** example). For this new app, **shopper**, we will be using **Arrow Function** syntax. Refactoring the code above to use an Arrow Function instead requires just a couple of steps: remove the word "function" and place an arrow (=>) between the argument and opening body bracket ({). Finally, we assign the Arrow Function (which is a JavaScript expression) to the variable **App** (formally the function name). Notice that in both cases, we **export App** (with the default keyword) so it can be imported into other files.

```
import React from 'react';
import './App.css';
```

```
const App = () => {
```

```
}
```

```
export default App;
```

Inside our Arrow Function we return **JSX** as follows. Notice that the **div** with a **class** of **App** is the

Intro to Frontend Development with React

Building App Components

single parent returned. Inside this **div** we return two sibling elements. The first is the **Nav** Component that we will create in the next section. The second is plain HTML markup containing a

span tag with the word “Empty” as a placeholder so we can validate that our initial code setup is working.

```
import React from 'react';
import Nav from './Nav';
import './App.css';

const App = () => {
  return (
    <div className="App">
      <Nav/>
      <main className="App-content">
        <span>Empty</span>
      </main>
    </div>
  );
}
```

Notice two important things with the code above. First, since we are using the **Nav** Component inside our **App** Component we must **import** it at the top of the file. Second, our **App** Component does not need the **ReactDOM** function from React as we used in the **HelloWorld** example. This is because our **exported App** Component is **imported** into the main **index.js** file which already contains the **render** logic as shown below.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

Nav Component

Next, we build the **Nav** Component. Open the **Nav.js** file and **import React** as we do for all React Components. Set up the **Nav** Component using the Arrow Function syntax and **export** it so we can **import** it into the **App** Component created in the previous section.

```
import React from 'react';

const Nav = () => {

}

export default Nav;
```

Inside the Arrow Function we return **JSX** with a parent of **nav** with a **class** of **App-nav**. Inside this element we return an Unordered List (****) with two sibling List Item (****) elements containing **buttons** representing the **Items** and **Cart** tabs. Each of these list elements are given class names of **App-nav-item** (this will allow us to target these elements for CSS styling). Notice that unlike plain HTML, **JSX** is inside JavaScript and since **class** is a JavaScript keyword we instead represent **class** inside **JSX** as **className**.

```
import React from 'react';

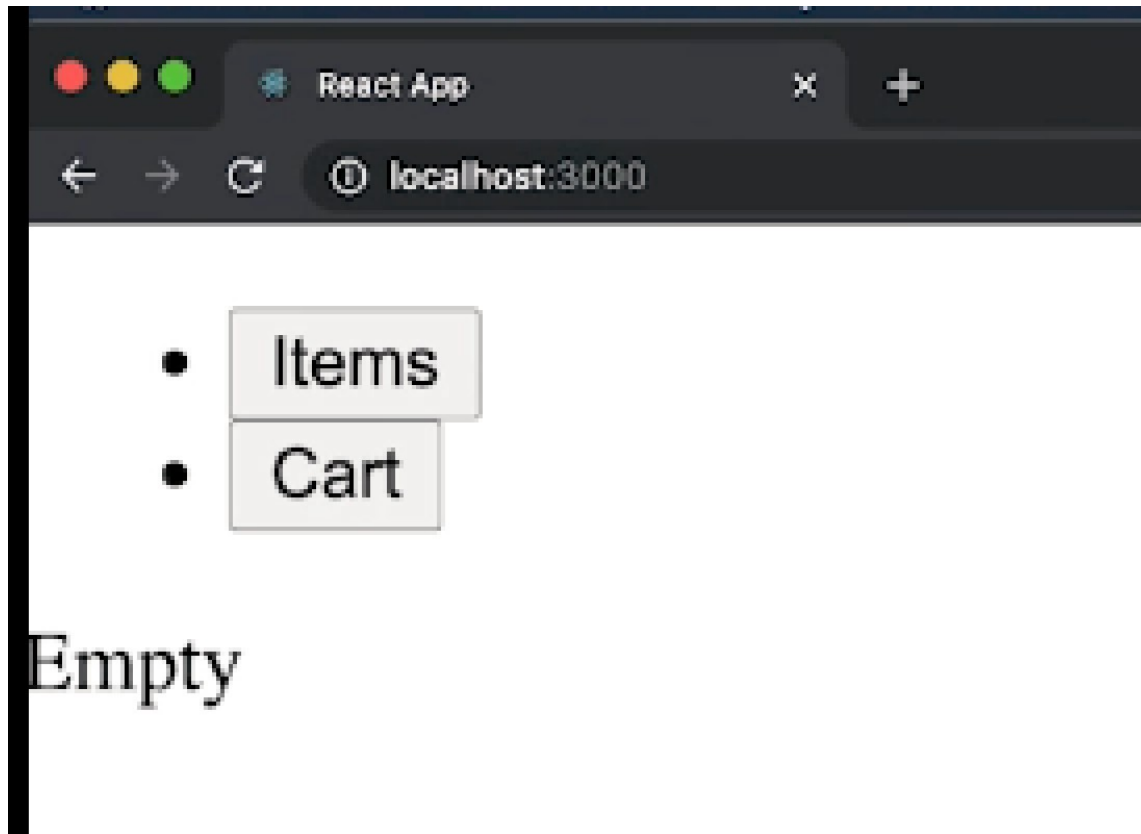
const Nav = () => {
  return (
    <nav className="App-nav">
      <ul>
        <li className="App-nav-item">
          <button>Items</button>
        </li>
        <li className="App-nav-item">
          <button>Cart</button>
        </li>
      </ul>
    </nav>
  )
}

export default Nav;
```

Test App in Browser

If your app is still running refresh the web page. In not, **start** it with the following command in the **terminal**. This command starts our React app and **displays** it in the **browser** (localhost:3000 by default).

```
npm start
```



In the last lesson, we started coding the **App** and **Nav** Components. In this lesson we will use the **class** names we provided to target our Component elements from a **CSS stylesheet**.

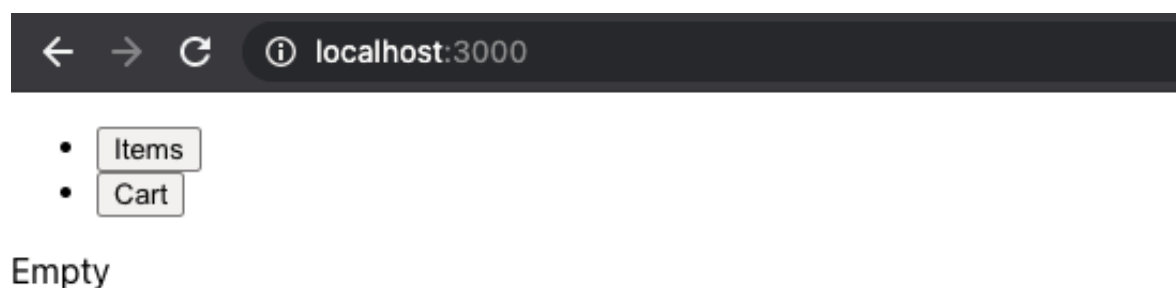
Applying CSS Component Style

To style our **App** Component, open the file **App.css** and ensure all its contents have been cleared out. Since we imported **App.css** into **App.js** any **CSS styles** we place in this file will affect the Component's **JSX** elements including other included Components (in this case **Nav**). From the **CSS stylesheet** (in this case **App.css**) we can **select** the **JSX elements** in the same manner we would normally select CSS. That is, to target an element by **id** in **CSS** we use the number sign (**#**) followed by the **id name** (**#className**). To target a **class** in **CSS** we use a **period** followed by the **class name** (**.App-nav**). To target a **tag** we use the **tag name** (**ul**). Finally, to target an element **within** a **class** we include both the **class name** as well as the **tag name** (**.App-nav ul**) separated by a **space**.

CSS Class – App Component

We will use this container class to set **margin** and a **max-width** within our web page. The **max-width** ensures our app grows no larger than the given pixel value even if the browser window is expanded more. The **margin** sets the space between HTML elements. In this case we center our app by setting the **top** and **bottom** to be zero pixels with the **left** and **right margin** automatically adjusted to keep our app horizontally centered on the page.

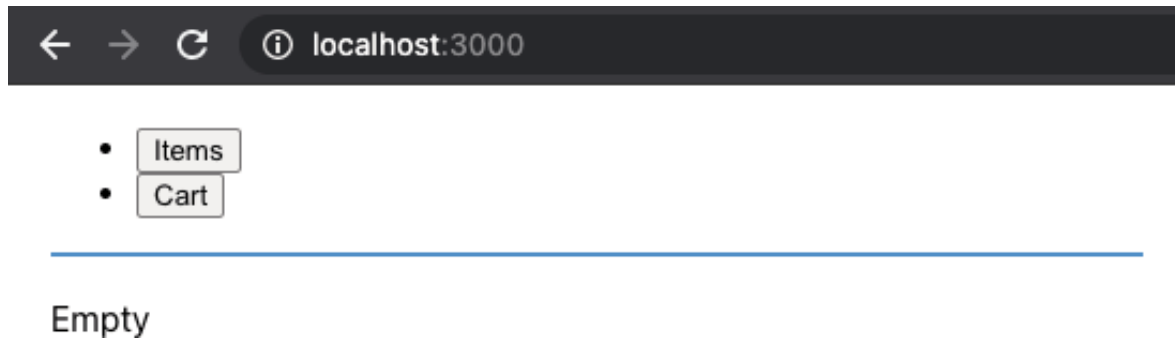
```
.App {  
  max-width: 800px;  
  margin: 0 auto;  
}
```



CSS Class – Nav Component

Within **App.css** we can also target the **Nav** Component because it is a child of our **App** Component. We will use its **App-nav** container class to set **margin**, **padding** and **border** on our navigation bar. The **margin** sets the space between HTML elements. The **padding** sets the space between a HTML element and its own contents. Finally the border provides lines (in this case just on bottom). This provides the solid line under the **Nav** Component we have in our design mockup.

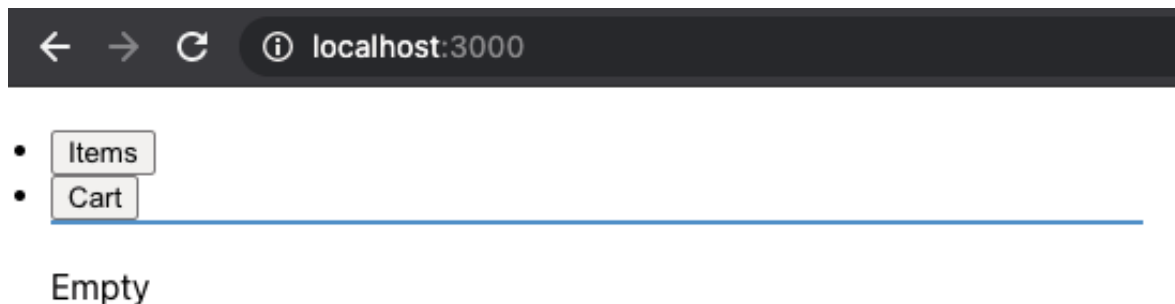
```
.App-nav {  
  margin: 20px;  
  padding: 18 px;  
  border-bottom: 2px solid #508fca;  
}
```



CSS Class – Nav List

Continuing with the **Nav** Component. We will use the CSS selector **.App-nav ul** to target the Unordered List (****) inside the **App Nav** container to reset **margin** and **padding** to zero.

```
.App-nav ul {  
  padding: 0;  
  margin: 0;  
}
```



CSS Class – Nav Buttons

Continuing with the **Nav** Component. We target the buttons in our **Nav** that represent our **Items** and **Cart** tabs. We also set the **font-size** for button text, take away any **background** by making it transparent, remove the default **border** that comes with a button, and finally change the cursor to by a **pointer** any time the mouse is over the button.

```
.App-nav button {  
  font-size: 18px;  
  background: transparent;  
  border: none;  
  cursor: pointer;  
}
```



- Items
- Cart

Empty

CSS Class – Nav List Items

Continuing with the **Nav** Component. We target each List Item (``) so that it no longer looks like a list but rather side-by-side buttons representing our **Items** and **Cart** tabs. We remove the default **list-style** (for example, the bullet points), set the **display** to place the buttons side by side, set the **font-size** and create a **border** on **bottom** that we will make visible when it's selected. We set its default color to be transparent.

```
.App-nav-item {  
  list-style: none;  
  display: inline-block;  
  margin-right: 32px;  
  font-size: 24px;  
  border-bottom: 4px solid transparent;  
}
```



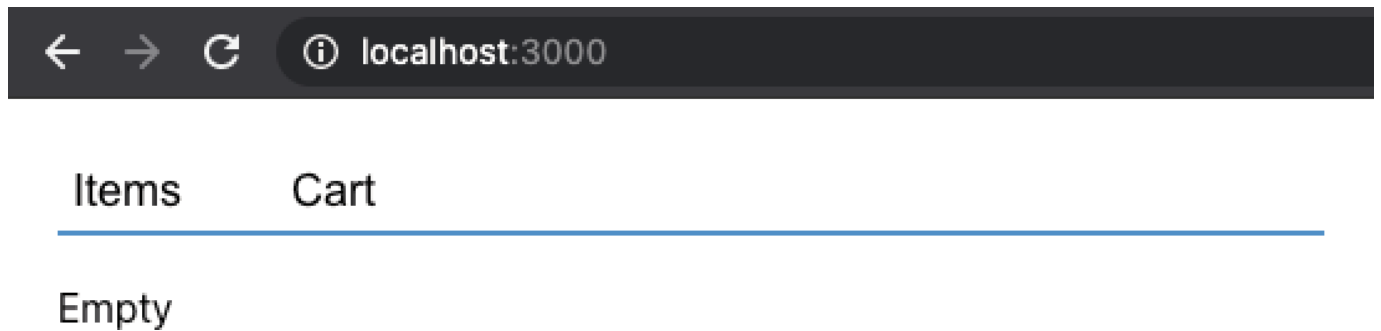
Items Cart

Empty

CSS Class – App Content

Our **App** currently contains a navigation bar on top and the rest of the app under the **Nav**. We want there to be no space between the **Nav** and the **App-content** but we do want horizontal space on the **left** and **right** our **App-content** container so we set **margin** to be **zero** on **top** and **bottom** with **left** and **right** set to **20 pixels**.

```
.App-content {  
  margin: 0 20px;  
}
```



Our **CSS stylesheet**, **App.css**, now looks like this. In future lessons we will add additional styles for interaction effect.

```
.App {  
  max-width: 800px;  
  margin: 0 auto;  
}  
  
.App-nav {  
  margin: 20px;  
  padding: 18px;  
  border-bottom: 2px solid #508fca;  
}  
  
.App-nav ul {  
  padding: 0;  
  margin: 0;  
}  
  
.App-nav button {  
  font-size: 18px;  
  background: transparent;  
  border: none;  
  cursor: pointer;  
}  
  
.App-nav-item {  
  list-style: none;  
  display: inline-block;  
  margin-right: 32px;  
  font-size: 24px;  
}  
  
.App-content {  
  margin: 0 20px;  
}
```

Intro to Frontend Development with React

Styling App Components

```
border-bottom: 4px solid transparent;  
}
```

```
.App-content {  
  margin: 0 20px;  
}
```

In this lesson we look at **React Hooks**. In particular we look at the **useState** Hook which allows to easily **access** and **update state** from within **React Functional Components**. We will use this functionality to keep track of the **active tab**; either **Items** or **Cart**. Knowing which **tab** is **active** allows us to display the correct page; either **ItemPage** or **CartPage**.

Anatomy of useState Hook

The **useState** Hook is a React function that takes the **initial state** as an **argument** and **returns** an **array** with **two** elements. The first element is the **current state** and the second element is a **function** which is used to **update** that **state**. The **useState** Hook is typically initialized using JavaScript **destructuring** which allows **initial state** to be set and the **state variable** and **update function** to be created all in one line at the top of the function.

```
const [activeTab, setActiveTab] = useState('items');
```

Rules of Hooks

React has a few rules for the usage of **Hooks**.

Rules of React Hooks

Only call hooks at the top level of your React Function Component

Only call hooks from React Function Components or custom hooks

The names of all hooks including custom hooks must start with “use” (example: useState)

Add Nav State using Hooks

The first thing we will use **hooks** for is to keep track of the **active tab**; either **Items** or **Cart**. Knowing which **tab** is **active** allows us to display the correct page; either **ItemPage** or **CartPage**. Inside our **App.js** file, import the **useState** function from React. Notice it is imported with curly braces. This is because **useState** is not the **default export** of the React library (**React** is – which is why it does not have braces around it). The next step is to add the initialization code at the top of the **App** Component setting the **state** variable to be named **activeTab** and its **update function** to be named **setActiveTab** with the initial **activeTab** state variable set to “items”.

```
// modified code
import React, {useState} from 'react';
// existing code
import Nav from './Nav';
import './App.css';

// modified code
const App = () => {
  // new code
  const [activeTab, setActiveTab] = useState('items');
  // existing code
  return (
    <div className="App">
      <Nav/>
      <main className="App-content">
        <span>Empty</span>
      </main>
    </div>
```

);


```
}  
  
// existing code  
export default App;
```

Pass State to Nav Component

Since we want our **Nav** Component to not only know the **activeTab state** but also be able to **update** it we pass both **activeTab** and **setActiveTab** to the Nav Component as **props**. We will update the **Nav** Component to use these props in the next lesson.

```
import React, {useState} from 'react';  
import Nav from './Nav';  
import './App.css';  
  
// modified code  
const App = () => {  
  // existing code  
  const [activeTab, setActiveTab] = useState('items');  
  return (  
    <div className="App">  
      {/* modified code */}  
      <Nav  
        activeTab={activeTab}  
        onChange={setActiveTab}  
      />  
      {/* existing code */}  
      <main className="App-content">  
        <span>Empty</span>  
      </main>  
    </div>  
  );  
}  
// existing code  
export default App;
```

Create App Content Component

Our **App-content** is currently just a placeholder. Inside the **App** Component, let's replace the **span** with the text "Empty" with a **Content** Component passing it the **activeTab state** as a **prop** named **tab**. Next, create the **Content** Component using the **Arrow Function** syntax inside our **App.js** file below the existing **App** Component. The **Content** Component accepts a prop **tab** (inside curly braces – JavaScript destructuring). We then use a JavaScript **switch** statement to return different content based on whether the **Items** or **Cart** tab is **currently selected**.

```
import React, {useState} from 'react';  
import Nav from './Nav';  
import './App.css';
```

Intro to Frontend Development with React

The Magic of Hooks - Part 1

```
// modified code
const App = () => {
```

Intro to Frontend Development with React

The Magic of Hooks - Part 1

```
// existing code
const [activeTab,
  setActiveTab] = useState('items');
return (
  <div className="App">
    {/* existing code */}
    <Nav activeTab={activeTab} onTabChange={setActiveTab}/>
    {/* modified code */}
    <main className="App-content">
      <Content tab={activeTab}/>
    </main>
  </div>
);
};

// new code
const Content = ({tab}) => {
  switch (tab) {
    case 'items':
      return <span>the items</span>;
    case 'cart':
      return <span>the cart</span>;
    default:
      break;
  }
};

// existing code
export default App;
```

In this lesson we continue working with **React Hooks**. In particular we use the **variable** and **update function** created in the **App** Component, using the **useState Hook**, and passed to the **Nav** Component as **props**. Within the **Nav** Component we create a **custom CSS class function** to add the **class name** "selected" to the **active tab** so it can be **styled in CSS**.

Current Nav Component

Currently we have a basic **Nav** Component that returns **JSX** with a parent of **nav** with a **class** of **App-nav**. Inside this element we return an Unordered List (****) with two sibling List Item (****) elements containing **buttons** representing the **Items** and **Cart** tabs. Each of these list elements are given **class names** of **App-nav-item** (which we style from **App.css**). Recall that since **class** is a JavaScript keyword we instead represent **class** inside **JSX** as **className**.

```
import React from 'react';

const Nav = () => {
  return (
    <nav className="App-nav">
      <ul>
        <li className="App-nav-item">
          <button>Items</button>
        </li>
        <li className="App-nav-item">
          <button>Cart</button>
        </li>
      </ul>
    </nav>
  )
}

export default Nav;
```

Modify Nav Component

The the first modification we make is enable the **Nav** Component to accept **props** (properties). Using JavaScript **destructuring** (by using the curly braces around variable names) we accept two props; **activeTab** and **onTabChange**. Recall that **activeTab** is the current **state** holding which **tab** is currently **active** and **onTabChange** is the **state update function** from the **useState Hook** which allows us to update state. Although these were both defined in the **App** Component we can use them in the **Nav** Component because they were passed in as **props**.

```
// existing code
import React from 'react';

// modified code
const Nav = ({activeTab, onTabChange}) => {
  return (
    <nav className="App-nav">
      <ul>
        <li className="App-nav-item">
          <button>Items</button>
```

Intro to Frontend Development with React

The Magic of Hooks - Part 2

```
</li>
```

```
<li className="App-nav-item">
```

```
        <button>Cart</button>
      </li>
    </ul>
  </nav>
)
}
// existing code
export default Nav;
```

Create Custom Class Names Function

We create a dynamic class names function that we can use inside **JSX** in place of a String class name. To work inside JSX, it must be placed inside curly braces which indicates to JSX that we will be using JavaScript here.

```
// existing code
import React from 'react';

// modified code
const Nav = ({activeTab, onTabChange}) => {
  // new code
  const itemClass = (tabName) => `
    App-nav-item ${
      (activeTab === tabName) ? 'selected' : ''
    }
  `;
  // existing code
  return (
    <nav className="App-nav">
      <ul>
        <li className="App-nav-item">
          <button>Items</button>
        </li>
        <li className="App-nav-item">
          <button>Cart</button>
        </li>
      </ul>
    </nav>
  )
}
// existing code
export default Nav;
```

Refactor JSX Class Names to use Function

Inside the **JSX** returned from the **Nav** Component modify the **className** to use our newly created **itemClass** function. Next, **modify** the **buttons** that represent our **Items** and **Cart tabs** to have a **click handler** that calls the **useState Hook update function** passed in as a prop, **onTabChange**,

Intro to Frontend Development with React

The Magic of Hooks - Part 2

passing the argument that represents its tab; the String “items” for the first button and “cart” for the second.

```
// existing code
```

```
import React from 'react';

// modified code
const Nav = ({activeTab, onTabChange}) => {
  // existing code
  const itemClass = (tabName) => `
    App-nav-item ${
      (activeTab === tabName) ? 'selected' : ''
    }
  `;

  // modified code
  return (
    <nav className="App-nav">
      <ul>
        {/* modified code */}
        <li className={itemClass('items')}>
          {/* modified code */}
          <button onClick={() => onTabChange('items')}>Items</button>
        </li>
        {/* modified code */}
        <li className={itemClass('cart')}>
          {/* modified code */}
          <button onClick={() => onTabChange('cart')}>Cart</button>
        </li>
      </ul>
    </nav>
  )
};

// existing code
export default Nav;
```

Update CSS for Tab Interactivity

In order to provide visual cues to the user to indicate the **selected tab** as well as when a **tab** is **hovered** over we will add two **CSS selectors** and corresponding styles inside **App.css** which is where we have all our **Nav** Component **CSS**. The first CSS selector (**.App-nav-item button: hover**) will **change** the **color** of the **tab** when it is **hovered** over. The second CSS selector (**.App-nav-item.selected**) will **change** the **color** of the **bottom border** of the **tab** to indicate which tab is **currently selected**.

```
/* existing code */
.App {
  max-width: 800px;
  margin: 0 auto;
}

.App-nav {
  margin: 20px;
  padding: 18 px;
  border-bottom: 2px solid #508fca;
```


Intro to Frontend Development with React

The Magic of Hooks - Part 2

```
}
```

```
.App-nav ul {
```

Intro to Frontend Development with React

The Magic of Hooks - Part 2

```
padding: 0;
margin: 0;
}

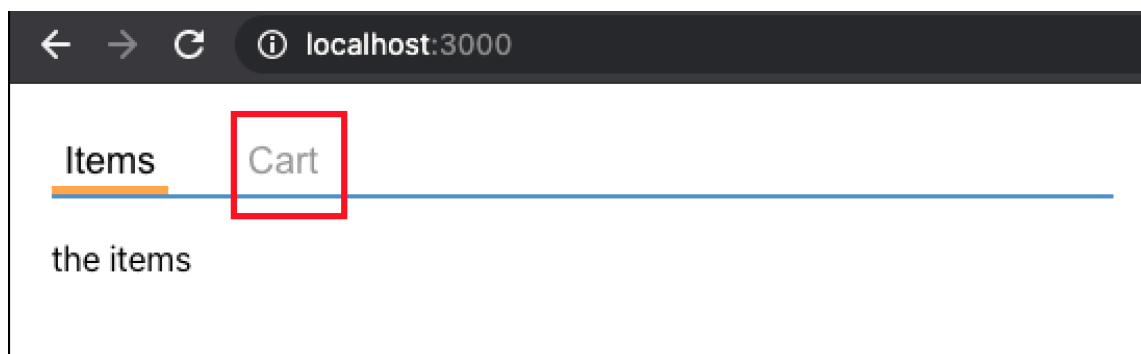
.App-nav button {
  font-size: 18px;
  background: transparent;
  border: none;
  cursor: pointer;
}

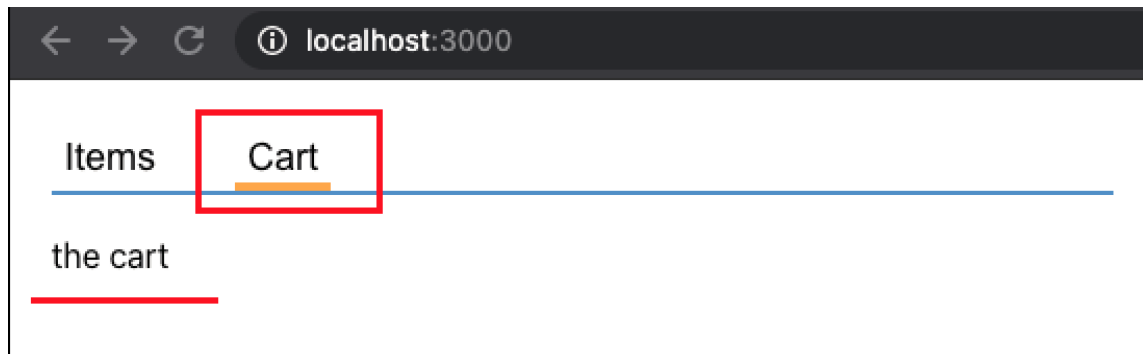
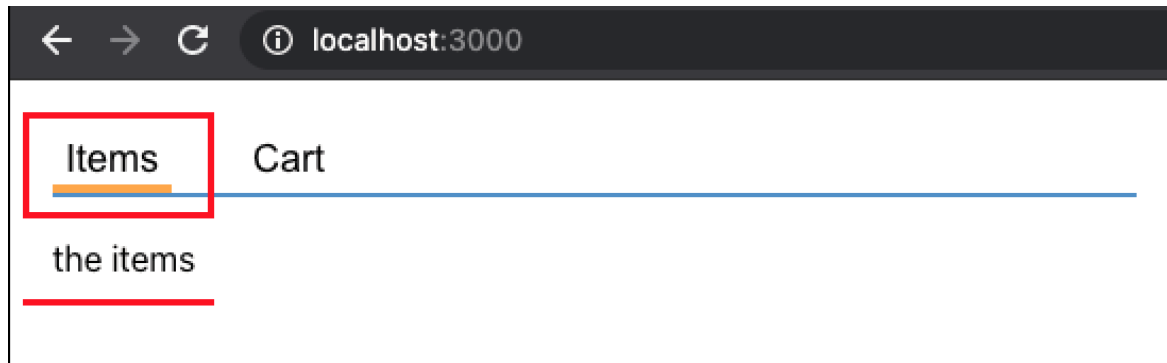
.App-nav-item {
  list-style: none;
  display: inline-block;
  margin-right: 32px;
  font-size: 24px;
  border-bottom: 4px solid transparent;
}

/* new code */
.App-nav-item button:hover {
  color: #999999;
}

/* new code */
.App-nav-item.selected {
  border-bottom-color: #ffaa3f;
}

/* existing code */
.App-content {
  margin: 0 20px;
}
```





In this lesson we will code the **ItemPage** Component while learning about two important React concepts: **Props** and **PropTypes**.

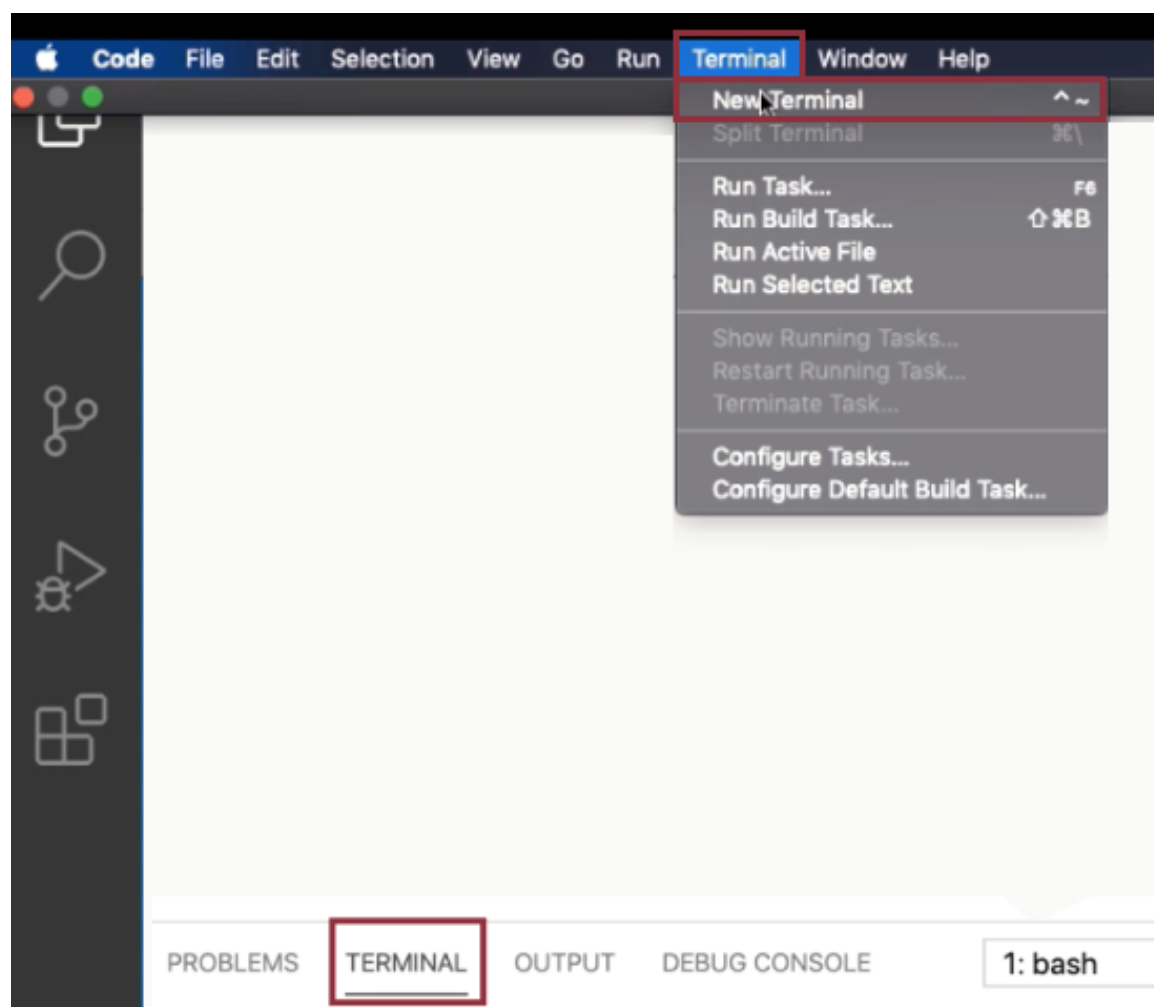
Props

Props in React Components are similar to **attributes** on HTML elements. In the last lesson we used **props** to pass a **variable** and a **function** to our **Nav** Component. What would happen if we passed the wrong data type or no argument when one was required? If there is data required and not available then JavaScript may throw a run-time error which not a good situation. One way to help avoid this is to use **PropTypes** which can help catch Prop-related errors at development time.

PropTypes

PropTypes are a **library** that helps in minimizing this problem in React by **checking** the **types** passed in the **props object** against a **specification** we set beforehand. Now, if the types passed don't match the types expected a warning is raised at development time. To **install PropTypes** into our project, open the **terminal** in the **project folder** and run the following command. In VS Code, you can use the **menu Terminal | New Terminal** to open the **integrated terminal** which always opens to the correct project folder.

```
npm install prop-types
```



ItemPage Component

In VS Code, open the file **ItemPage.js** that we created previously. Let's start by importing the **React library**. Next, let's import the **PropTypes** library we installed in the previous section. The final **import** is our **ItemPage.css** stylesheet file (which we will create in the next lesson). Next, we create the **ItemPage** Component as a regular function which accepts one parameter, **items**, which is an array passed in as a **Prop** using JavaScript destructuring with the curly braces. Finally we export the **ItemPage** Component (with the default keyword) so it can be imported into other files.

```
import React from 'react';
import PropTypes from 'prop-types';
import './ItemPage.css';

function ItemPage({items}) {

}

export default ItemPage;
```

ItemPage Component – JSX

Inside the **ItemPage** Component we return **JSX** with a parent Unordered List (****) with a **class** of **ItemPage-items**. Inside this list element we write **JavaScript code** (inside curly braces) to **map** (loop through an array returning a new array) over the **items** Prop. Inside the JavaScript **map** function we use an **Arrow Function** to accept each **item** in the **items** array and return a List Item (****) element with a **unique key** (required) and a **CSS class** (**ItemPage-item**). In order to test that everything is working so far we simply return from **JSX** a list of the **item names**.

```
import React from 'react';
import PropTypes from 'prop-types';
import './ItemPage.css';

function ItemPage({items}) {
  return (
    <ul className="ItemPage-items">
      {items.map(item =>
        <li key={item.id} className="ItemPage-item">
          {item.name}
        </li>)}
    </ul>
  );
}

export default ItemPage;
```

ItemPage Component – PropTypes

Next, just below the **ItemPage** Component, we will set up **PropTypes** for the **ItemPage** Component. We declare PropTypes as an **object property** on the Component. The **properties**

Intro to Frontend Development with React

Props and Proptypes - Part 1

inside this **object** are the **Props** and the **values** are **PropType Validators**. In our case, we have just one; the array **items**.

Intro to Frontend Development with React

Props and Proptypes - Part 1

```
// existing code
import React from 'react';
import PropTypes from 'prop-types';
import './ItemPage.css';

// existing code
function ItemPage({items}) {
  return (
    <ul className="ItemPage-items">
      {items.map(item =>
        <li key={item.id} className="ItemPage-item">
          {item.name}
        </li>)}
    </ul>
  );
}

// new code
ItemPage.propTypes = {
  items: PropTypes.array.isRequired
};

// existing code
export default ItemPage;
```

In the last lesson we coded our **ItemPage** Component. In this lesson we complete the code, mock static-data, and add CSS styling for the **ItemPage** Component. Our **ItemPage** JavaScript code currently looks like this. Notice the **CSS classes** that we will be adding style for: **ItemPage-items** (****) and **ItemPage-item** (****).

```
import React from 'react';
import PropTypes from 'prop-types';
import './ItemPage.css';

function ItemPage({items}) {
  return (
    <ul className="ItemPage-items">
      {items.map(item =>
        <li key={item.id} className="ItemPage-item">
          {item.name}
        </li>)}
    </ul>
  );
}

ItemPage.propTypes = {
  items: PropTypes.array.isRequired
};

export default ItemPage;
```

ItemPage – CSS Styling

Create a new **file** in the **src** folder with all the Component and CSS files called **ItemPage.css** (we already have an **import** statement in the **ItemPage.js** to use this styling). The **ItemPage** Component requires some basic styles for **ItemPage-items** (****) and **ItemPage-item** (****). First, for the **** we remove its default **margin** and **padding**. Second, for the **** we remove its default style (bullet points when inside a **** tag) and add some **margin** on **bottom** to provide space between each item in the list.

```
.ItemPage-items {
  margin: 0;
  padding: 0;
}

.ItemPage-item {
  list-style: none;
  margin-bottom: 20px;
}
```

ItemPage – Create Test Data

To better design and test our app we use static data often called mock or test data. **Create** a new

Intro to Frontend Development with React

Props and Proptypes - Part 2

file in the **src** folder with all the Component and CSS files called **static-data.js** (notice the **.js** file extension).

```
let items = [
  {
    id: 0,
    name: "Apple iPad",
    description: "An iPad like no other.",
    price: 329.00
  },
  {
    id: 1,
    name: "Apple iPad Pro",
    description: "Even more expensive than the iPad.",
    price: 729.00
  },
];

export {items};
```

App Component – Include ItemPage Component

To use the **ItemPage** Component and the **static-data** we just created **import** the **Component** and the exported **items** array into the **App** Component. Then, in the **Content** Component (included in the **App.js** file) we can change the **switch case** of “items” to **return** the **ItemPage** Component instead of the test **span** we had in place. Recall that when we created the **ItemPage** Component in the last lesson we set it up to **accept** a **prop** called **items**. We can now **pass** the **static-data** to the **ItemPage** Component as a **prop** called **items**. Notice that **items** is **imported** inside curly braces. This is because it was an **export** not a **default export** from the file **static-data.js**

```
// existing code
import React, {useState} from 'react';
import Nav from './Nav';
import './App.css';
// new code
import ItemPage from './ItemPage';
import {items} from './static-data';

// existing code
const App = () => {
  const [activeTab,
    setActiveTab] = useState('items');

  return (
    <div className="App">
      <Nav activeTab={activeTab} onTabChange={setActiveTab}/>
      <main className="App-content">
        <Content tab={activeTab}/>
      </main>
    </div>
  );
};
```

Intro to Frontend Development with React

Props and Proptypes - Part 2

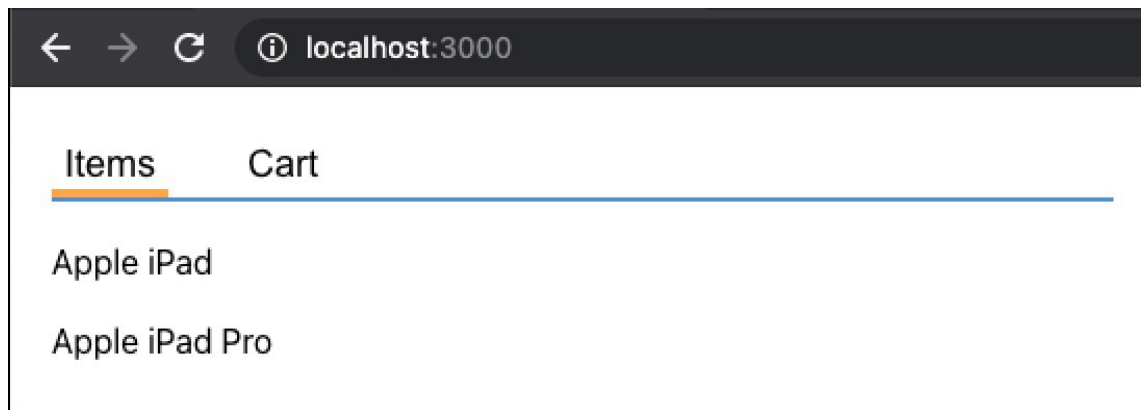
```
// modified code
```

```
const Content = ({tab}) => {  
  switch (tab) {
```

```
    case 'items':
      // modified code
      return <ItemPage items={items}/>;
    case 'cart':
      return <span>the cart</span>;
    default:
      break;
  }
};

// existing code
export default App;
```

Our **web page** now shows our **imported** test **items**.



In this lesson we will code the **Item** Component including **PropTypes** for **data-type** and **is-required** validation of the Component's **Props**. As a refresher, this image shows the **left** and **right** areas we will be coding. In the next lesson we work on the display and styling.



Item Component

In VS Code, open the file **Item.js** that we created previously. Let's start by **importing** the **React** and **PropTypes** libraries. Next, we add an **import** for the **Item.css** stylesheet file (which we will create in the next lesson). Next, we create the **Item** Component using **Arrow Function** syntax which we set up to accept two parameters, **item** and **onAddToCart**, which are passed in as **Props** using JavaScript destructuring with the curly braces. Finally we export the **Item** Component (with the default keyword) so it can be imported into other files.

```
import React from 'react';
import PropTypes from 'prop-types';
import './Item.css';

const Item = ({item, onAddToCart}) => (

)

export default Item;
```

Item Component – JSX

Inside the **Item** Component we return **JSX** with a parent container (**<div>**) with a **class** of **Item**. Inside this **outer** container we set up **two inner** containers; (**<div>**) with a **class** of **Item-left** and **Item-right**. On the **left** side we include several **HTML** elements with **classes** of **Item-image**, **Item-title**, and **Item-description**. Recall from our **static-data** that each **item** has **properties** which we can now include inside each **<div>** (**id**, **name**, **description**, **price**). On the **right** side we include two **HTML** elements. The first is a **<div>** with a **class** of **Item-price** to hold our **price** property and the second, a **HTML button** with a **class** of **Item-addToCart** and the text “Add to Cart”.

```
// existing code
import React from 'react';
import PropTypes from 'prop-types';
import './Item.css';
```

```
// modified code
const Item = ({item, onAddToCart}) => (
  // new code
  <div className="Item">
    <div className="Item-left">
      <div className="Item-image"></div>
      <div className="Item-title">
        {item.name}
      </div>
      <div className="Item-description">
        {item.description}
      </div>
    </div>
    <div className="Item-right">
      <div className="Item-price">
        ${item.price}
      </div>
      <button className="Item-addToCart">
        Add to Cart
      </button>
    </div>
  </div>
)
// existing code
export default Item;
```

Item Component – PropTypes

Inside the **Item** Component we also add **PropTypes** for **data-type** and **is-required** validation of the Component's **Props**. The first **Prop**, **item**, is a required Object. The second **Prop**, **onAddToCart**, is a required Function.

```
// existing code
import React from 'react';
import PropTypes from 'prop-types';
import './Item.css';

// existing code
const Item = ({item, onAddToCart}) => {
  <div className="Item">
    <div className="Item-left">
      <div className="Item-image"></div>
      <div className="Item-title">
        {item.name}
      </div>
      <div className="Item-description">
        {item.description}
      </div>
    </div>
    <div className="Item-right">
```

Intro to Frontend Development with React

Create Item Component

```
<div className="Item-price">
  ${item.price}
</div>
```

```
    <button className="Item-addToCart">
      Add to Cart
    </button>
  </div>
</div>
}

// new code
Item.propTypes = {
  item: PropTypes.object.isRequired,
  onAddToCart: PropTypes.func.isRequired
};

// existing code
export default Item;
```


In this lesson we implement the **display logic** and **CSS styling** for our **Item** Component. The image below, from our design, shows the result of our **Item** Component **display** and **styling**.



ItemPage Component – Include Item Component

To use the **Item** Component we **import** it into the **ItemPage** Component (**ItemPage.js**). Next, we modify the **ItemPage** Component **Props** to include the **onAddToCart** function (received from **App** and used by **Item**). We will modify the **App** Component in the next section. Within the **ItemPage** Component, refactor the **contents** of the **** with a **class** of **ItemPage-item** from containing the **item name** to be our newly imported **Item** Component. We can then pass the current **item** and **onAddToCart** function as **Props** to the **Item** Component. Notice the value passed to the **onAddToCart Prop** is an **Arrow Function** with **current item** passed to it.

```
// existing code
import React from 'react';
import PropTypes from 'prop-types';
import './ItemPage.css';
// new code
import Item from './Item';

// modified code
function ItemPage({items, onAddToCart}) {
  return (
    <ul className="ItemPage-items">
      {items.map(item =>
        <li key={item.id} className="ItemPage-item">
          {/* modified code */}
          <Item item={item} onAddToCart={() => onAddToCart(item)} />
        </li>)}
    </ul>
  );
}

// existing code
ItemPage.propTypes = {
  items: PropTypes.array.isRequired
};

// existing code
export default ItemPage;
```

App Component – Create Cart State

Since our **ItemPage** Component now requires a **Prop** function called **onAddToCart**, we must pass this from our **App** Component where we include **ItemPage**. To make this work we require several updates inside the **App** Component (**App.js** file). First, we create a new **state** variable (**cart**) and **state update** function (with **useState** Hook) initializing the **cart state** to an **empty array** (**[]**).

```
// existing code
import React, {useState} from 'react';
import Nav from './Nav';
import './App.css';
import ItemPage from './ItemPage';
import {items} from './static-data';

// modified code
const App = () => {
  // existing code
  const [activeTab, setActiveTab] = useState('items');
  // new code
  const [cart, setCart] = useState([]);

  // existing code
  return (
    <div className="App">
      <Nav activeTab={activeTab} onTabChange={setActiveTab}/>
      <main className="App-content">
        <Content tab={activeTab}/>
      </main>
    </div>
  );
};

// existing code
const Content = ({tab}) => {
  switch (tab) {
    case 'items':
      return <ItemPage items={items}/>;
    case 'cart':
      return <span>the cart</span>;
    default:
      break;
  }
};

// existing code
export default App;
```

App Component – Modify ItemPage Props

We need to modify the **Props** we are passing to the **ItemPage** Component to include an

Intro to Frontend Development with React

Displaying and Styling Item

onAddToCart function as an additional **Prop**. In the previous section we initialized the **cart** state and its **update** function. In this section we will first **create** a new function, **addToCart**, that will be responsible for calling the **cart** state **update** function. Notice that we call **setCart** passing it an

array created by using the JavaScript **spread operator** to include all current array items and add an additional array element which is the **current item** passed in to the function. We can then pass this function, **addToCart**, to the **Content** Component as the additional **Prop** named **onAddToCart** which will then be forwarded to the **ItemPage** Component and finally to the **Item** Component where it will be used. Notice how Props flow through your app Components but only in one direction (parent to child).

```
// existing code
import React, {useState} from 'react';
import Nav from './Nav';
import './App.css';
import ItemPage from './ItemPage';
import {items} from './static-data';

// modified code
const App = () => {
  // existing code
  const [activeTab, setActiveTab] = useState('items');
  const [cart, setCart] = useState([]);

  // new code
  const addToCart = (item) => {
    setCart([...cart, item]);
  };

  // modified code
  return (
    <div className="App">
      <Nav activeTab={activeTab} onTabChange={setActiveTab}/>
      <main className="App-content">
        {/* modified code (added onAddToCart) */}
        <Content tab={activeTab} onAddToCart={addToCart}/>
      </main>
    </div>
  );
};

// modified code (added onAddToCart)
const Content = ({tab, onAddToCart}) => {
  switch (tab) {
    case 'items':
      // modified code (added onAddToCart)
      return <ItemPage items={items} onAddToCart={onAddToCart}/>;
    case 'cart':
      return <span>the cart</span>;
    default:
      break;
  }
};
```

Intro to Frontend Development with React

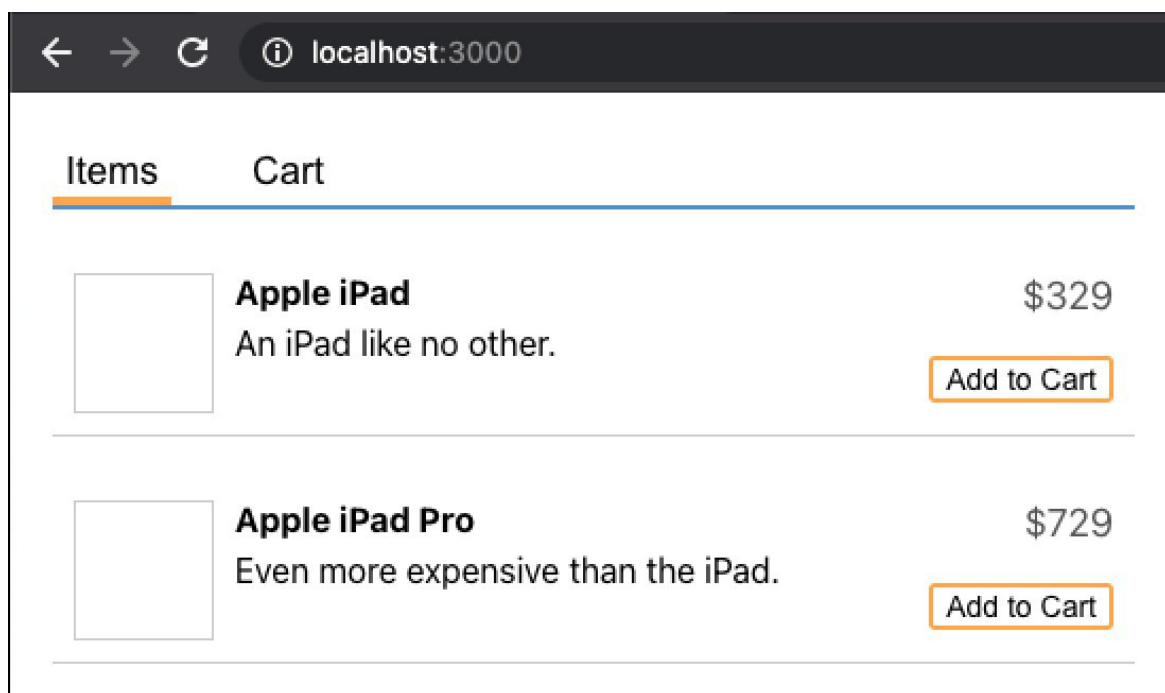
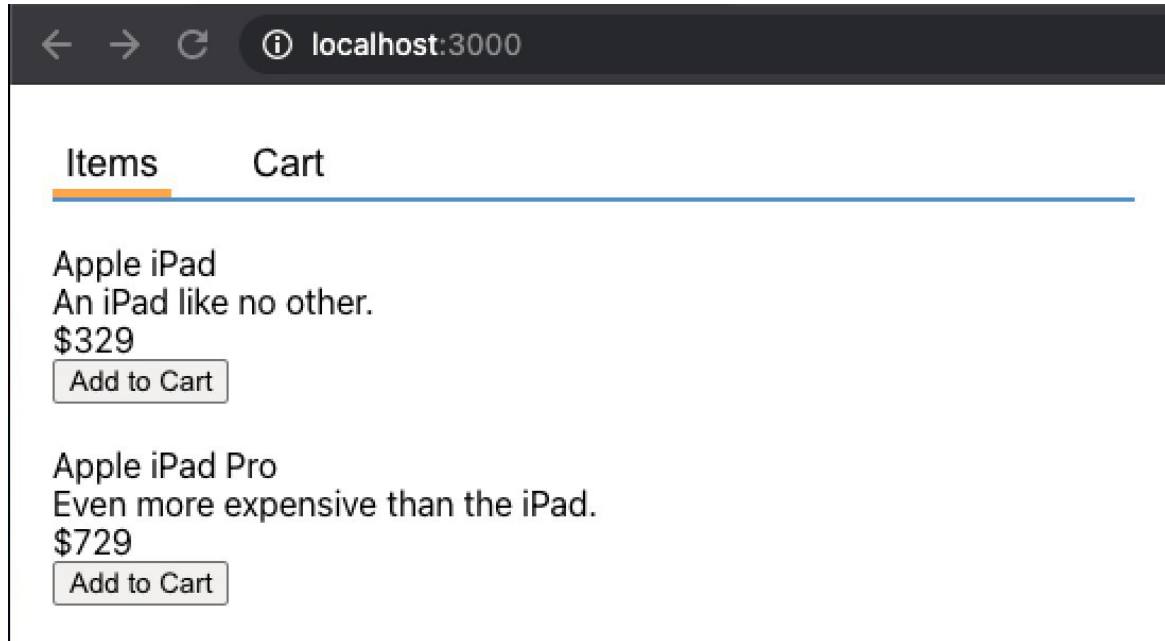
Displaying and Styling Item

```
// existing code
```

```
export default App;
```

Item Component – CSS Styling

It's now time to provide the CSS **styling** to all the CSS **classes** we used in the **Item** Component. The two images that follow show the difference CSS makes to our webpage. The first image is before styling the **Item** Component and the second image is the same content with CSS styling applied.



Create a new **file** in the **src** folder with all the Component and CSS files called **Item.css** (we already have an **import** statement in the **Item.js** to use this styling). Since this course focuses on React and we have already discussed the CSS Styling of several components, you have the option of copying the **Item** CSS from the course downloads or from the code block below and moving onto the next lesson. However, since a basic understanding of CSS is so fundamental to any web development the next section illustrates step by step how our **CSS** transforms the **Item** Component **between** the **before** and **after** images shown above. Here is the complete CSS for **Item**

Component.

```
.Item {
  display: flex;
  justify-content: space-between;
  border-bottom: 1px solid #ccc;
  padding: 10px;
}
.Item-image {
  width: 64px;
  height: 64px;
  border: 1px solid #ccc;
  margin-right: 10px;
  float: left;
}
.Item-title {
  font-weight: bold;
  margin-bottom: 0.4em;
}
.Item-price {
  text-align: right;
  font-size: 18px;
  color: #555;
}
.Item-addToCart {
  margin-bottom: 5px;
  font-size: 14px;
  border: 2px solid #FFAA3F;
  background-color: #fff;
  border-radius: 3px;
  cursor: pointer;
}
.Item-addToCart:hover {
  background-color: #FFDDB2;
}
.Item-addToCart:active {
  background-color: #ED8400;
  color: #fff;
  outline: none;
}
.Item-addToCart:focus {
  outline: none;
}
.Item-left {
  flex: 1;
}
.Item-right {
  display: flex;
  flex-direction: column;
```

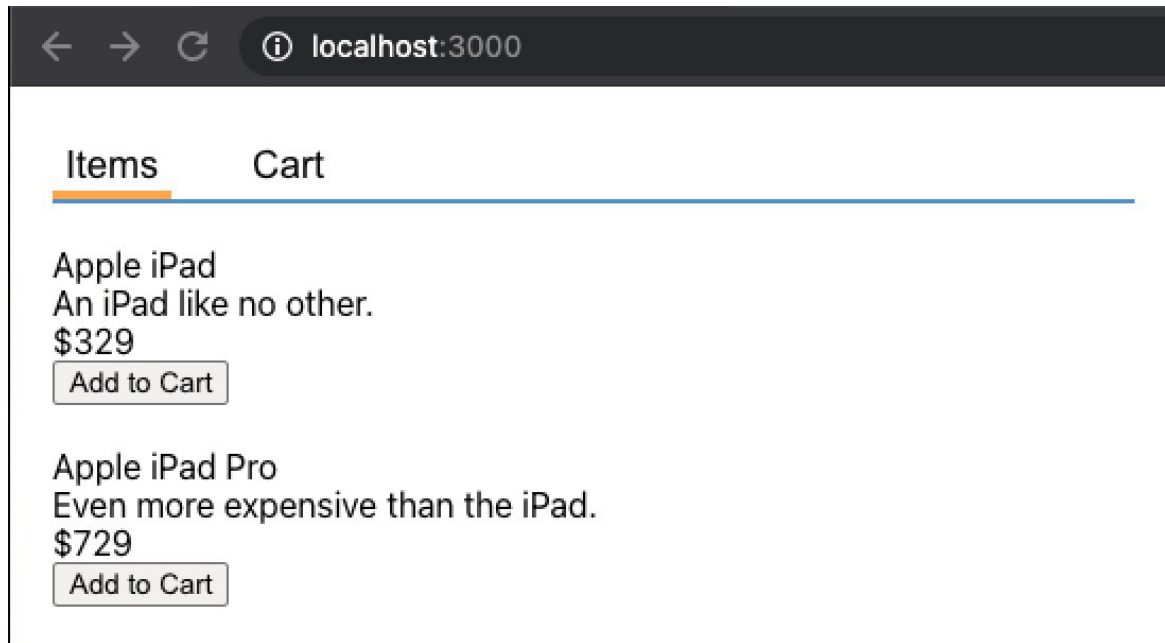
Intro to Frontend Development with React

Displaying and Styling Item

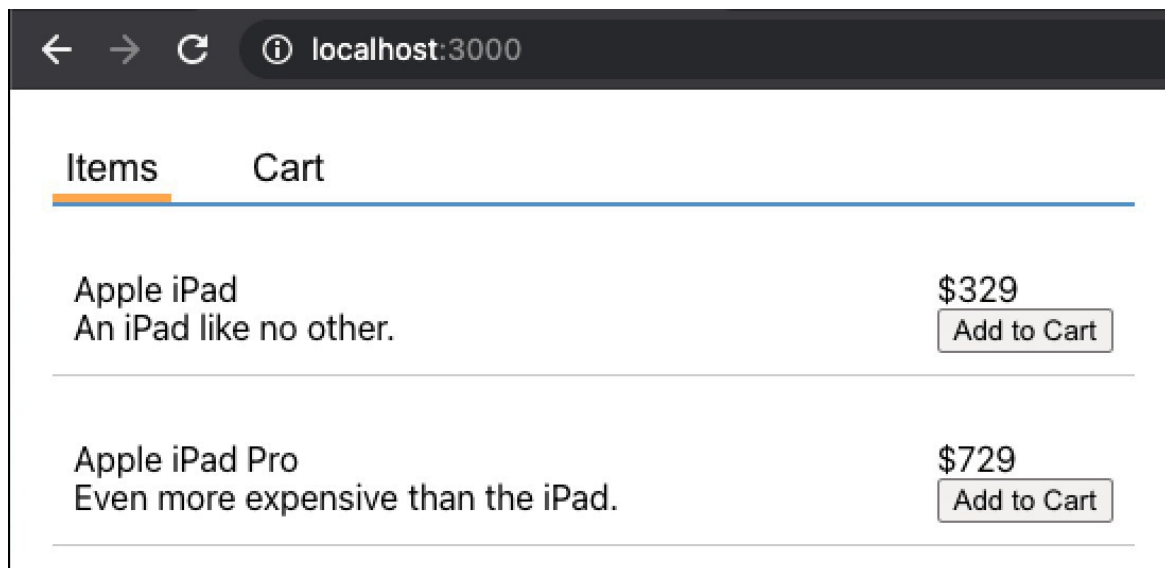
```
justify-content: space-between;
```

```
}
```

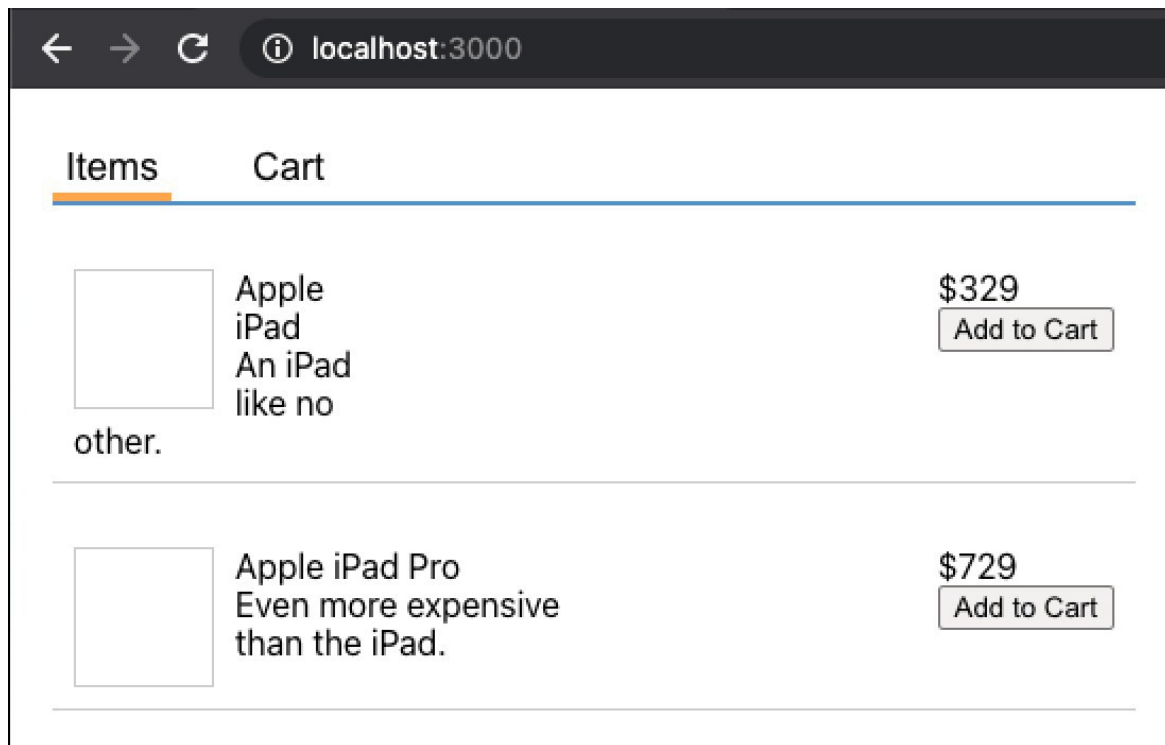
Item – Step by Step CSS Styling



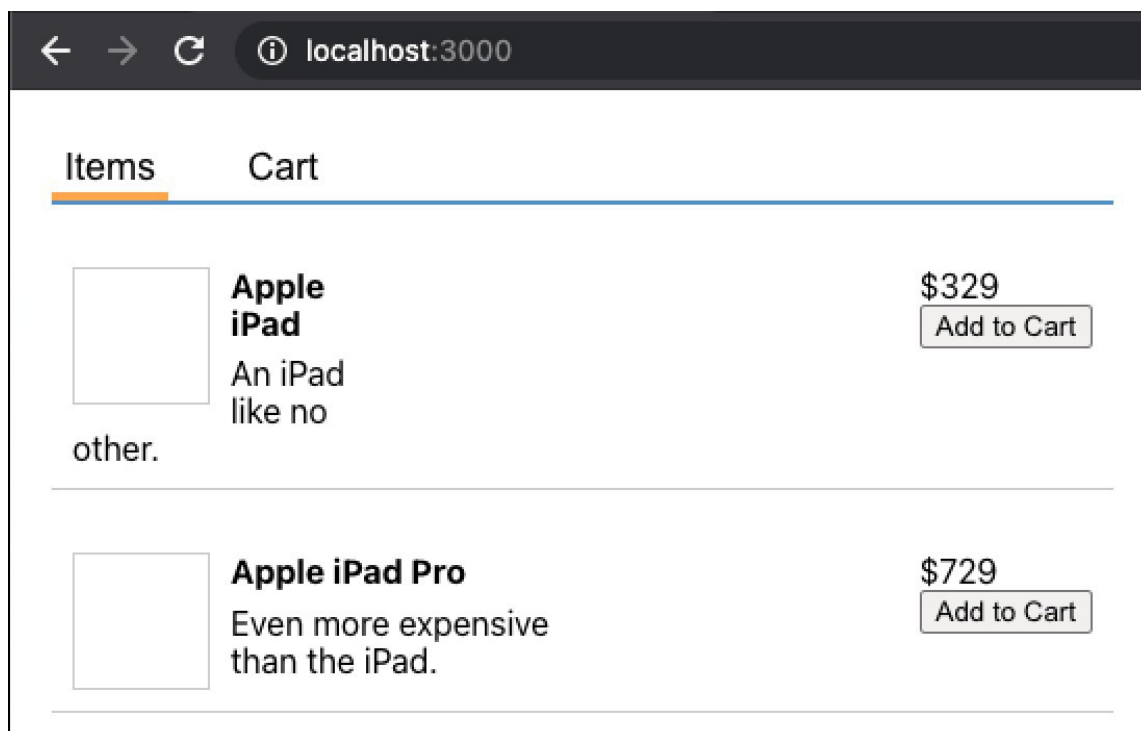
```
.Item {  
  display: flex;  
  justify-content: space-between;  
  border-bottom: 1px solid #ccc;  
  padding: 10px;  
}
```



```
.Item-image {  
  width: 64px;  
  height: 64px;  
  border: 1px solid #ccc;  
  margin-right: 10px;  
} float: left;
```



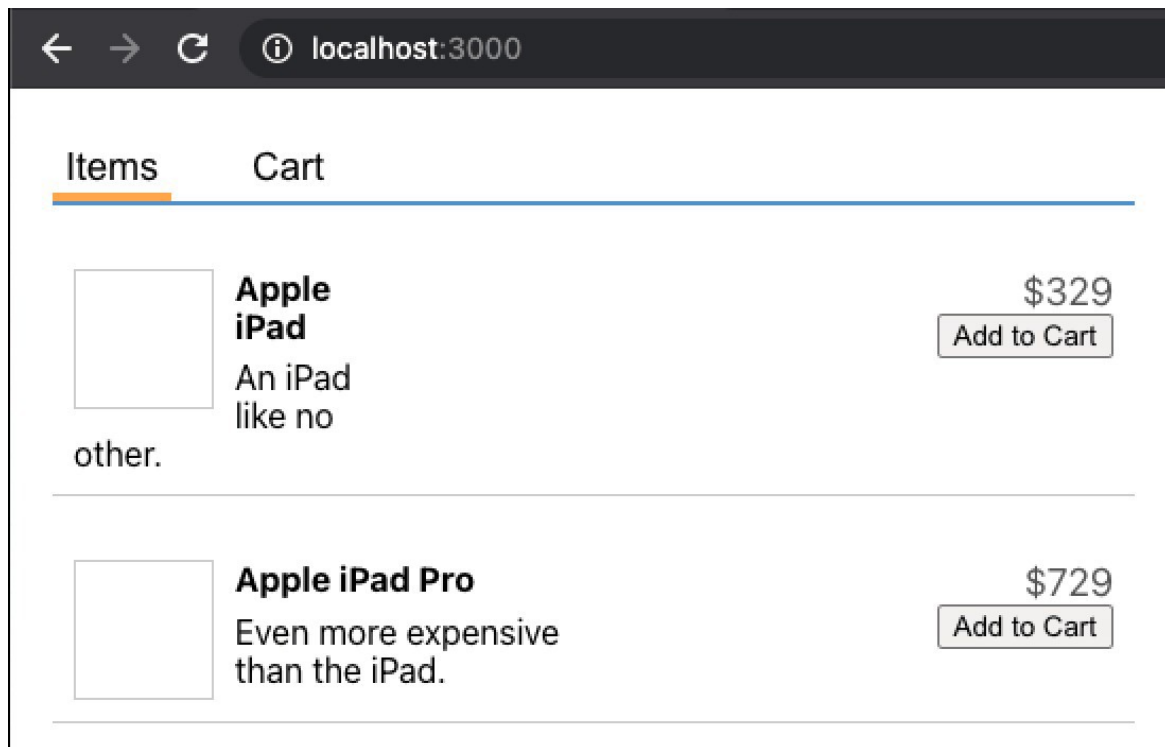
```
.Item-title {  
  font-weight: bold;  
  margin-bottom: 0.4em;  
}
```



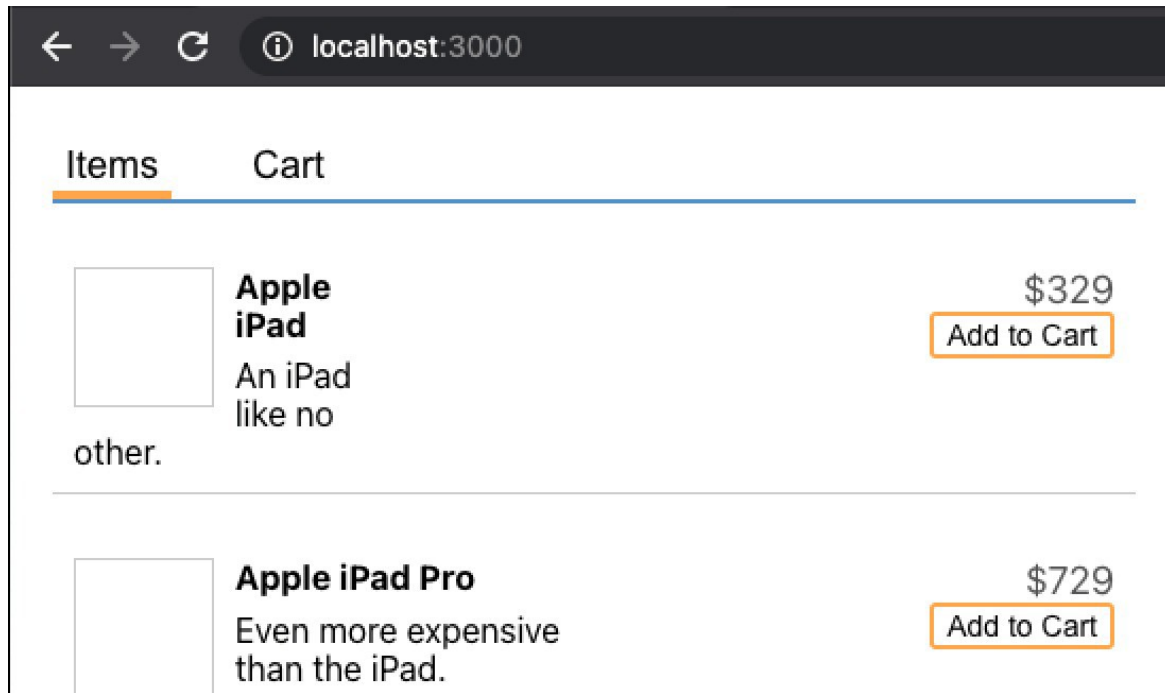
Intro to Frontend Development with React

Displaying and Styling Item

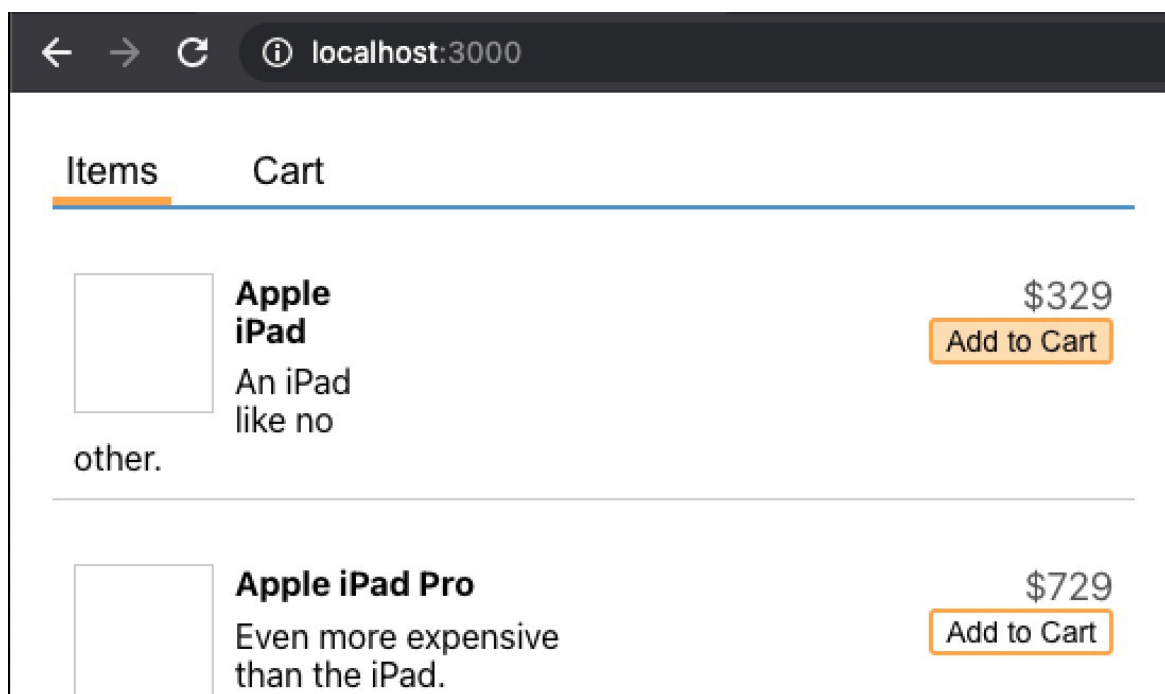
```
.Item-price {  
  text-align: right;  
  font-size: 18px;  
  color: #555;  
}
```



```
.Item-addToCart {  
  margin-bottom: 5px;  
  font-size: 14px;  
  border: 2px solid #FFAA3F;  
  background-color: #fff;  
  border-radius: 3px;  
  cursor: pointer;  
}
```



```
.Item-addToCart:hover {  
  background-color: #FFDDB2;  
}
```



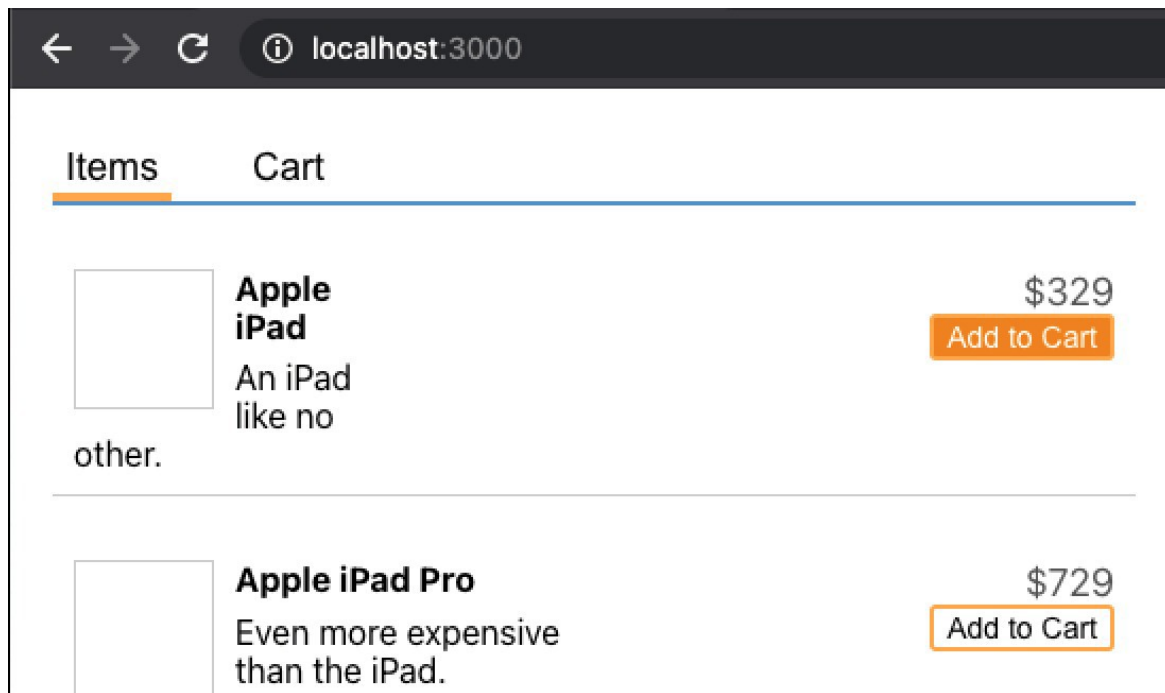
```
.Item-addToCart:active {  
  background-color: #ED8400;  
  color: #fff;  
}
```

Intro to Frontend Development with React

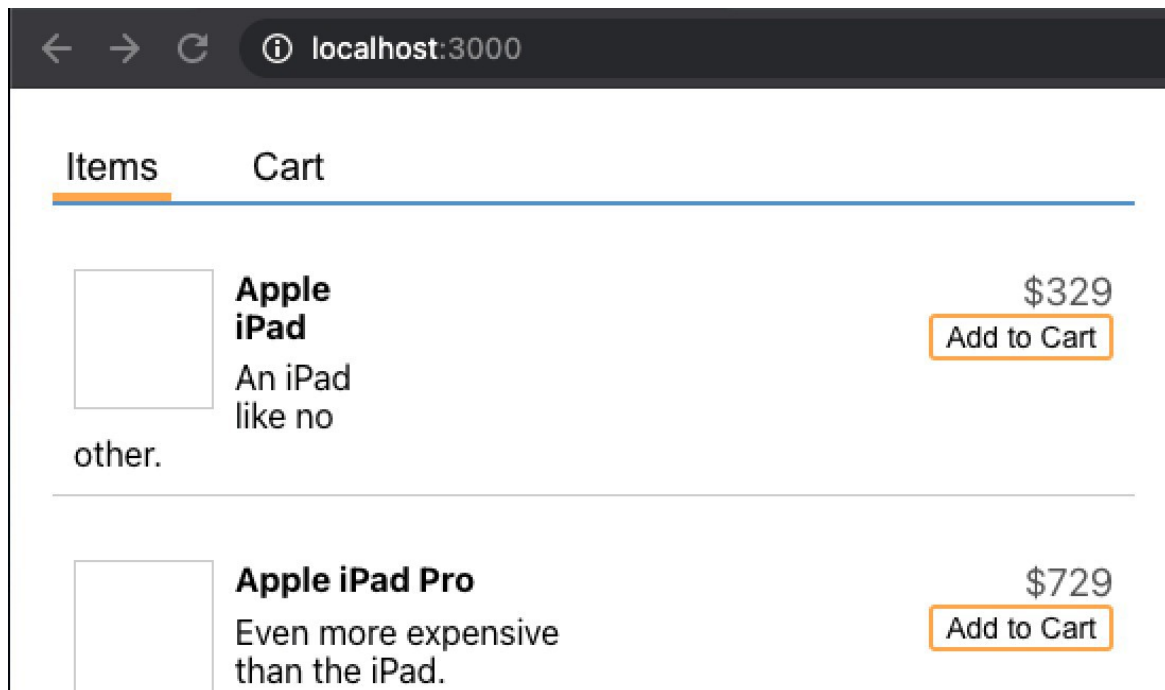
Displaying and Styling Item

```
outline: none;
```

```
}
```



```
.Item-addToCart:focus {  
  outline: none;  
}
```



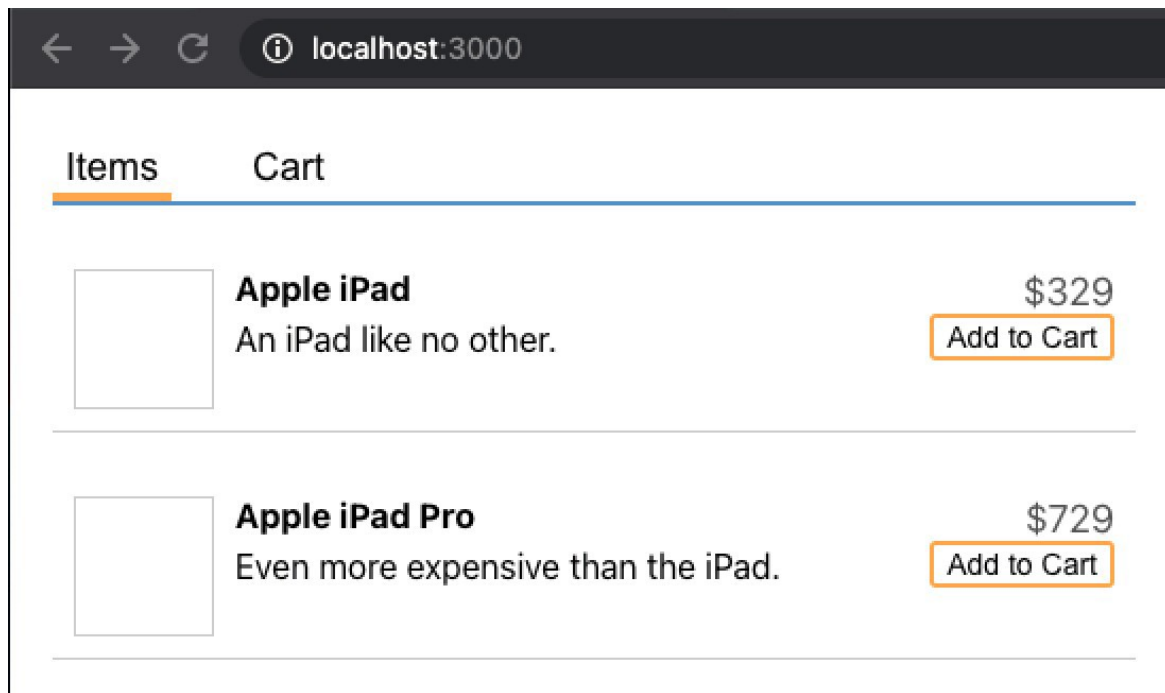
```
.Item-left {  
}
```

Intro to Frontend Development with React

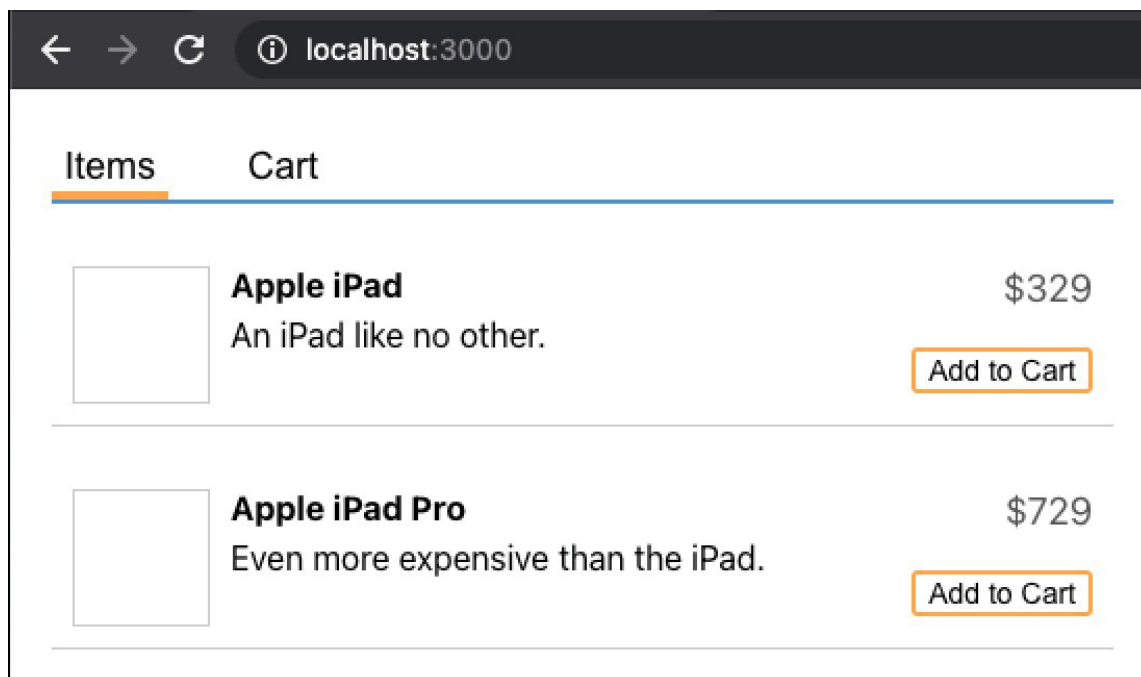
Displaying and Styling Item

```
flex: 1;
```

```
}
```



```
.Item-right {  
  display: flex;  
  flex-direction: column;  
  justify-content: space-between;  
}
```



In this lesson we create our final Component; **CartPage**. We started our e-commerce web app by using **Create-React-App**. This tool set up our base React project and provided us with starter code. We then refactored the **App** Component to be our new application parent and created four empty JavaScript files that would become our **Nav**, **Item**, **ItemPage** and **CartPage** Components. The first three we coded from scratch. For our final Component, **CartPage**, we will use a common developer practice of **copy | paste | refactor**.

CartPage Component

Since **CartPage** is similar to **ItemPage** we open **ItemPage.js** and **copy** all its code. Open the file **CartPage.js** that we created previously and **paste** the copied code. The next step is to refactor the **CartPage** Component. Start by changing all six occurrences of **ItemPage** to **CartPage**.

```
import React from 'react';
import PropTypes from 'prop-types';
import Item from './Item';
// modified code
import './CartPage.css';

// modified code (3 occurrences)
function CartPage({items, onAddToCart}) {
  return (
    <ul className="CartPage-items">
      {items.map(item =>
        <li key={item.id} className="CartPage-item">
          <Item item={item} onAddToCart={() => onAddToCart(item)}/>
        </li>)}
    </ul>
  );
}
// modified code
CartPage.propTypes = {
  items: PropTypes.array.isRequired
};
// modified code
export default CartPage;
```

CartPage – CSS Stylesheet

In the **CartPage** code we **refactored** above we **import** the **CartPage.css** file. **Create** this new file in the **src** folder with all the Component and CSS files. We will add the CSS styles in an upcoming lesson.

CartPage Component – Props

Next, we refactor the **CartPage** Component's **Props**. Since **CartPage** will not have an **AddToCart** button we can remove the **onAddToCart** Prop from the **function parameters** as well as from the **Props** passed to the **Item** Component.

```
// existing code
import React from 'react';
```

Intro to Frontend Development with React

Create CartPage

```
import PropTypes from 'prop-types';  
import Item from './Item';
```

```
import './CartPage.css';

// modified code (remove onAddToCart Prop)
function CartPage({items}) {
  return (
    <ul className="CartPage-items">
      {items.map(item =>
        <li key={item.id} className="CartPage-item">
          /* modified code (remove onAddToCart Prop) */
          <Item item={item}/>
        </li>)}
    </ul>
  );
}
// existing code
CartPage.propTypes = {
  items: PropTypes.array.isRequired
};
// existing code
export default CartPage;
```

App Component – Include CartPage Component

To use the **CartPage** Component we just created, **import** the **Component** into the **App** Component.

```
// existing code
import React, {useState} from 'react';
import Nav from './Nav';
import './App.css';
import {items} from './static-data';
import ItemPage from './ItemPage';
// new code
import CartPage from './CartPage';

// existing code(App Component)
const App = () => {
};

// existing code(Content Component)
const Content = ({tab, onAddToCart}) => {
};

// existing code
export default App;
```

App Component – Summarize Cart Items

Intro to Frontend Development with React

Create CartPage

The **CartPage** Component needs to display the items added to the **cart**. However, at this point, our **cart state** only contains **items array indices** not the actual list of **items**. On top of that there can be duplicate indices (when an item is added multiple times). To correct this we need to refactor the

App Component. Create a function we'll call **summarizeCart** that will group the **cart state items** by their **id** and keep **count** of how many of each **item** is in the **cart**. Modify **App.js** by creating the function above the **App** Component. Using an Arrow Function and the **Array REDUCE method** we can group the **cart items**.

```
// existing code (imports)

// new code
const summarizeCart = (cart) => {
  const groupItems = cart.reduce((summary, item) => {
    summary[item.id] = summary[item.id] || {
      ...item,
      count: 0
    }
    summary[item.id].count++;

    return summary;
  }, {});

  return Object.values(groupItems);
};

// existing code (App Component)
const App = () => {
};

// existing code (Content Component)
const Content = ({tab, onAddToCart}) => {
};

// existing code
export default App;
```

JavaScript's **Array REDUCE method** creates a **loop** with a **memory** which allows it to return a **single value** unlike its siblings (**map** | **sort** | **filter**) that return a **modified array**. The **reduce** method takes a **reducer** function which has **two arguments** (**summary**, **item**). The **summary** is the **previous value** (**running total**) while **item** is the **current cart** Object. As **cart.reduce()** loops through each **item** in the **cart state** it returns a value which becomes the new **summary** value. This new value is created if it does not exist (**item** properties plus a new **count** property **initialized** to **zero**). Once it is created the **count** property is **increased** by **one**. After all **items** have been **looped** through the **groupItems** is equal to the last **summary** value and is returned from the **summarizeCart** function. Notice that **cart.reduce()** takes a **second argument** which is the **initial value** of **summary** (we've set this to an empty Object). Another **important** factor to notice is the **return value** from the **summarizeCart** function. If we simply returned **groupItems** it would be an Object of Objects. Instead, by returning **Object.values(groupItems)** we get an Array of Objects like the original **items** from our **static-data**. The only difference is we now have an added **count** property. It's important that the format match since both **ItemPage** and **CartPage** use the same **Item** Component which expects the same Prop **items** with matching PropTypes.

App Component – Include Cart as Prop

Now that we have a way to create a **summarized cart**, we use this function inside the **App**

Component as the **value** for the **cart** Prop which we pass to the **Content** Component. Next, we update the **Content** Component to accept this new Prop and pass it to our newly included **CartPage** Component inside the **switch case** of "cart" in place of the **span** we currently have for testing.

```
// existing code
import React, {useState} from 'react';
import Nav from './Nav';
import './App.css';
import {items} from './static-data';
import ItemPage from './ItemPage';
import CartPage from './CartPage';

// existing code
const summarizeCart = (cart) => {
  const groupItems = cart.reduce((summary, item) => {
    summary[item.id] = summary[item.id] || {
      ...item,
      count: 0
    }
    summary[item.id].count++;

    return summary;
  }, {});

  return Object.values(groupItems);
};

// modified code
const App = () => {
  // existing code
  const [activeTab, setActiveTab] = useState('items');
  const [cart, setCart] = useState([]);

  const addToCart = (item) => {
    setCart([...cart, item]);
  };

  // modified code
  return (
    <div className="App">
      <Nav activeTab={activeTab} onTabChange={setActiveTab}/>
      <main className="App-content">
        {/* modified code (add cart Prop) */}
        <Content
          tab={activeTab}
          onAddToCart={addToCart}
          cart={summarizeCart(cart)}/>
      </main>
    </div>
  );
};
```

```
};
```

```
// modified code (and cart Prop)
```

```
const Content = ({tab, onAddToCart, cart}) => {
```

```
switch (tab) {
```

```
    case 'items':
      return <ItemPage items={items} onAddToCart={onAddToCart}/>;
    case 'cart':
      // modified code (add CartPage and items Prop set to cart)
      return <CartPage items={cart}/>
    default:
      break;
  }
};

// existing code
export default App;
```

Item Component – Children Prop

Now that we are using the **Item** Component from both the **ItemPage** and **CartPage** we have an issue. The **AddToCart** button appears on the **CartPage** because it is part of the **Item** Component. To fix this we refactor both the **Item** and **ItemPage** Components to **cut** the **AddToCart** button from **Item** and **paste** it within **ItemPage** replacing it within **Item** by a special Prop called **children**. The **children** Prop takes everything that is included between the opening and closing Component tags; in this case the **Item** Component. The **children** Prop replaces the **onAddToCart** Prop in the **Item** Component **function parameters**. The other change we need to make is to remove the **onAddToCart** Prop from **Item PropTypes**.

```
// existing code
import React from 'react';
import PropTypes from 'prop-types';
import './Item.css';

// modified code (replace onAddToCart with children Prop)
const Item = ({item, children}) => {
  return (
    <div className="Item">
      <div className="Item-left">
        <div className="Item-image"></div>
        <div className="Item-title">
          {item.name}
        </div>
        <div className="Item-description">
          {item.description}
        </div>
      </div>
      <div className="Item-right">
        <div className="Item-price">
          ${item.price}
        </div>
        {/* modified code (replace button with children Prop)*/}
        {children}
      </div>
    </div>
  );
};
```

Intro to Frontend Development with React

Create CartPage

```
</div>  
);  
}
```

```
// modified code (remove onAddToCart Prop)
Item.propTypes = {
  item: PropTypes.object.isRequired,
};

// existing code
export default Item;
```

ItemPage Component – AddToCart Button

Modify the **Item** Component inside **ItemPage** to have an opening and closing tag. In between these tags **paste** the **button** we just **cut** from the **Item** Component. One additional change is required. The **onAddToCart** Prop that was being passed to the **Item** Component can be removed with its **value** (the arrow function) becoming the **new value** of the button's **onClick** handler.

```
// existing code
import React from 'react';
import PropTypes from 'prop-types';
import './ItemPage.css';
import Item from './Item';

// modified code
function ItemPage({items, onAddToCart}) {
  return (
    <ul className="ItemPage-items">
      {items.map(item =>
        <li key={item.id} className="ItemPage-item">
          {/* modified code (added button inside Item) */}
          <Item item={item}>
            <button className="Item-addToCart" onClick={()=>onAddToCart(item)}>
              Add to Cart
            </button>
          </Item>
        </li>
      )}
    </ul>
  );
}

// existing code
ItemPage.propTypes = {
  items: PropTypes.array.isRequired,
  onAddToCart: PropTypes.func.isRequired
};

// existing code
export default ItemPage;
```

In this lesson we continue to code our final Component, **CartPage**, by adding the functionality to **view** and **adjust** the **quantity** of an **item** in the **cart**.

CartPage Component – CartItem Controls

In the last lesson we included the **AddToCart** button in the **ItemPage** inside the **Item** Component tags. This resulted in the **AddToCart** button being passed to the **Item** Component as a special Prop called **children**. In a similar way, we will now add **CartItem Controls** inside the **Item** Component tags of **CartPage** which will result in these controls being passed to the **Item** Component as **children**.

CartPage Component – Props and PropTypes

To add the functionality to **adjust** the **quantity** of an **item** in the **cart** we accept two new function Props to the **CartPage** Component; **onAddOne** and **onRemoveOne**. We will code and supply these functions to **CartPage** from **App** in the next section. With two new Props, we add them to the **CartPage** PropTypes as required functions.

```
// existing code (imports)

// modified code (add new Props)
function CartPage({items, onAddOne, onRemoveOne}) {
  return (
    <ul className="CartPage-items">
      {items.map(item =>
        <li key={item.id} className="CartPage-item">
          <Item item={item}>
            </Item>
          </li>)}
    </ul>
  );
}
// modified code (add new Props)
CartPage.propTypes = {
  items: PropTypes.array.isRequired,
  onAddOne: PropTypes.func.isRequired,
  onRemoveOne: PropTypes.func.isRequired
};
// existing code
export default CartPage;
```

Content Component – New Props for CartPage

In the previous section, we refactored **CartPage** to accept two new function Props; **onAddOne** and **onRemoveOne**. For **onAddOne**, we can just pass the existing **onAddToCart** Prop to **increase** the **quantity** in **cart**. For **onRemoveOne**, we will code a new function within the **App** Component to **remove** an **item** from the **cart** when **quantity** is **decreased**. First, **refactor** the **Content** Component in the **App.js** file to **accept** a new Prop, **onRemoveItem**, and pass Props **onAddOne** (value is **onAddToCart**) and **onRemoveOne** (value is **onRemoveItem**) to the **CartPage** Component.

Intro to Frontend Development with React

Adding Events to CartPage

```
// existing code (imports)
```

```
// existing code
const summarizeCart = (cart) => {
};

// existing code (App Component)
const App = () => {
};

// modified code (add onRemoveItem Prop)
const Content = ({tab, onAddToCart, cart, onRemoveItem}) => {
  switch (tab) {
    case 'items':
      return <ItemPage items={items} onAddToCart={onAddToCart}/>;
    case 'cart':
      // modified code (add onAddOne and onRemoveOne)
      return <CartPage items={cart} onAddOne={onAddToCart} onRemoveOne={onRemoveItem}
/>
    default:
      break;
  }
};

// existing code
export default App;
```

App Component – Remove Cart Items

In the previous section, we refactored **Content** to accept a new function Prop; **onRemoveItem**. We then passed that Prop as **value** to **CartPage** Prop **onRemoveOne**. In this section, we refactor the **App** Component to code the new function, **removeItem**. Just below our current **addToCart** function, create a new Arrow Function, **removeItem**, and use the **Array SPLICE** method to **remove** an **item** from the **cart array** and then update the **cart state**. First, we find the **index** of the **item** in the **cart**. Then inside the **cart state update** function, **setCart**, we take a **copy** of the **cart state** and use the **index** we found to **remove** the correct **item** before returning the **copy** as our new **cart state**.

```
// existing code (imports)

// existing code
const summarizeCart = (cart) => {
};

// modified code (App Component)
const App = () => {
  // existing code
  const [activeTab,
    setActiveTab] = useState('items');
  const [cart,
    setCart] = useState([]);
```


Intro to Frontend Development with React

Adding Events to CartPage

```
const addToCart = (item) => {  
  setCart([  
    ...cart,
```

```
        item
      });
    };

    // new code
    const removeItem = (item) => {
      let index = cart.findIndex(i => i.id === item.id);
      if (index >= 0) {
        setCart(cart => {
          const copy = [...cart];
          copy.splice(index, 1);
          return copy;
        });
      }
    }

    // existing code
    return (
      <div className="App">
        <Nav activeTab={activeTab} onTabChange={setActiveTab}/>
        <main className="App-content">
          <Content tab={activeTab} onAddToCart={addToCart} cart={summarizeCart(cart)}/>
        </main>
      </div>
    );
  };

  // existing code (Content Component)
  const Content = ({tab, onAddToCart, cart, onRemoveItem}) => {
  };

  // existing code
  export default App;
```

App Component – New Props for Content

In the previous section, we refactored **App** to have a new function, **removeItem**. Now, within the **App** Component **JSX** we can add the Prop, **onRemoveItem**, to **Content** setting its **value** to our new function, **removeItem**.

```
// existing code (imports)

// existing code
const summarizeCart = (cart) => {
};

// modified code (App Component)
const App = () => {
```

Intro to Frontend Development with React

Adding Events to CartPage

```
// existing code
```

```
const [activeTab,  
    setActiveTab] = useState('items');
```

```
const [cart,
```

```
    setCart] = useState([]);

const addToCart = (item) => {
  setCart([
    ...cart,
    item
  ]);
};

const removeItem = (item) => {
  let index = cart.findIndex(i => i.id === item.id);
  if (index >= 0) {
    setCart(cart => {
      const copy = [...cart];
      copy.splice(index, 1);
      return copy;
    });
  }
}

// modified code
return (
  <div className="App">
    <Nav activeTab={activeTab} onTabChange={setActiveTab}/>
    <main className="App-content">
      /* modified code (add onRemoveItem Prop) */
      <Content tab={activeTab}
        onAddToCart={addToCart}
        cart={summarizeCart(cart)}
        onRemoveItem={removeItem}/>
    </main>
  </div>
);
};

// existing code (Content Component)
const Content = ({tab, onAddToCart, cart, onRemoveItem}) => {
};

// existing code
export default App;
```

CartPage – Testing

With our app running in the browser we should now be able to, on the **ItemPage**, use the **AddToCart** button to add one or more **items** to the **cart**. When we switch to the **CartPage**, we should see the **items** we added along with the **quantity** we added. On **CartPage** there should also be **buttons** that **increase** and **decrease** the **quantity**. With our **CartPage functionality** in place,

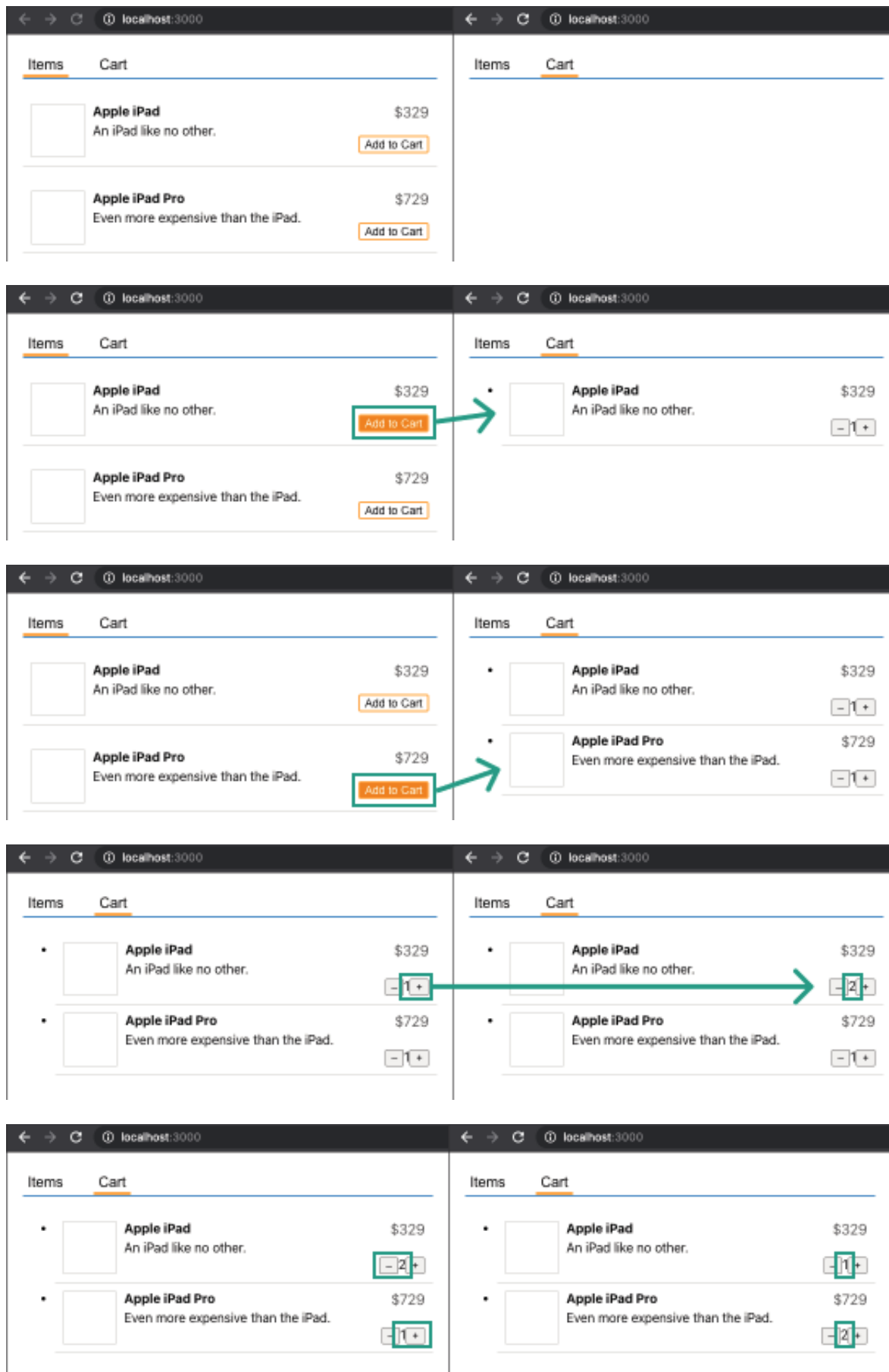
Intro to Frontend Development with React

Adding Events to CartPage

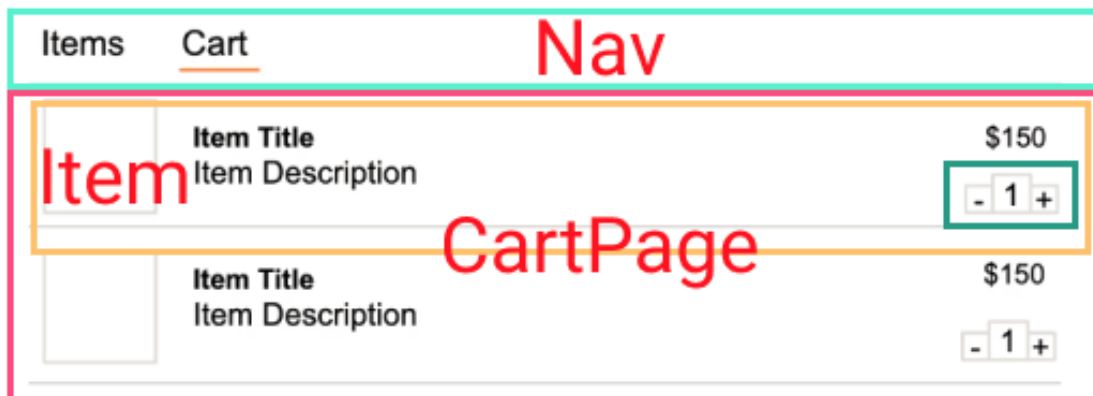
we will complete the **CartPage CSS styling** in the next lesson.

Intro to Frontend Development with React

Adding Events to CartPage

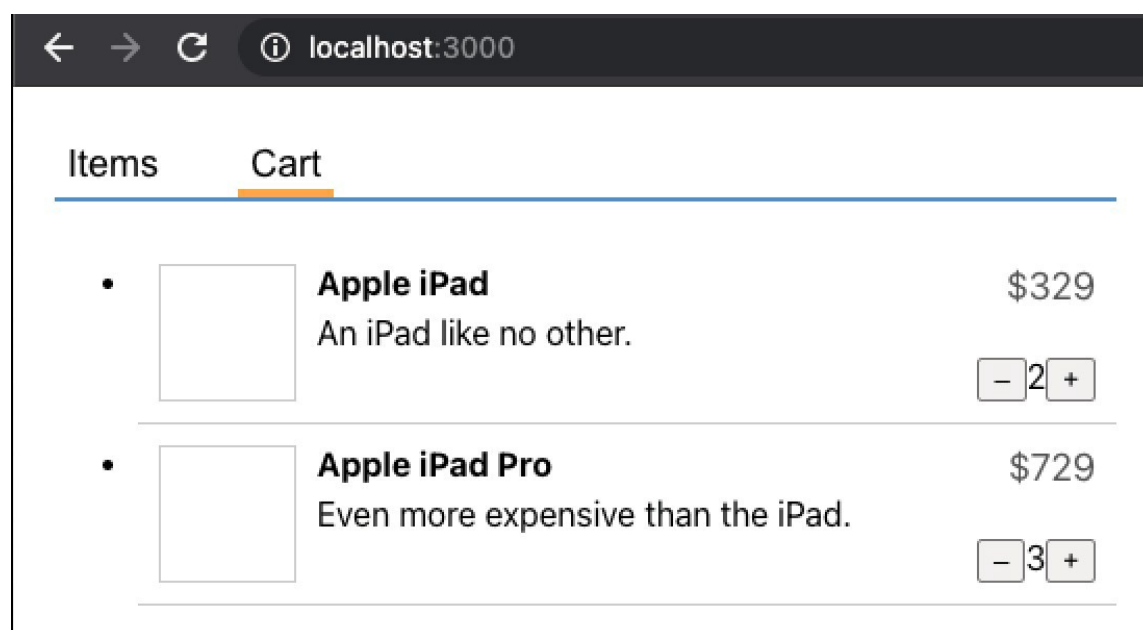


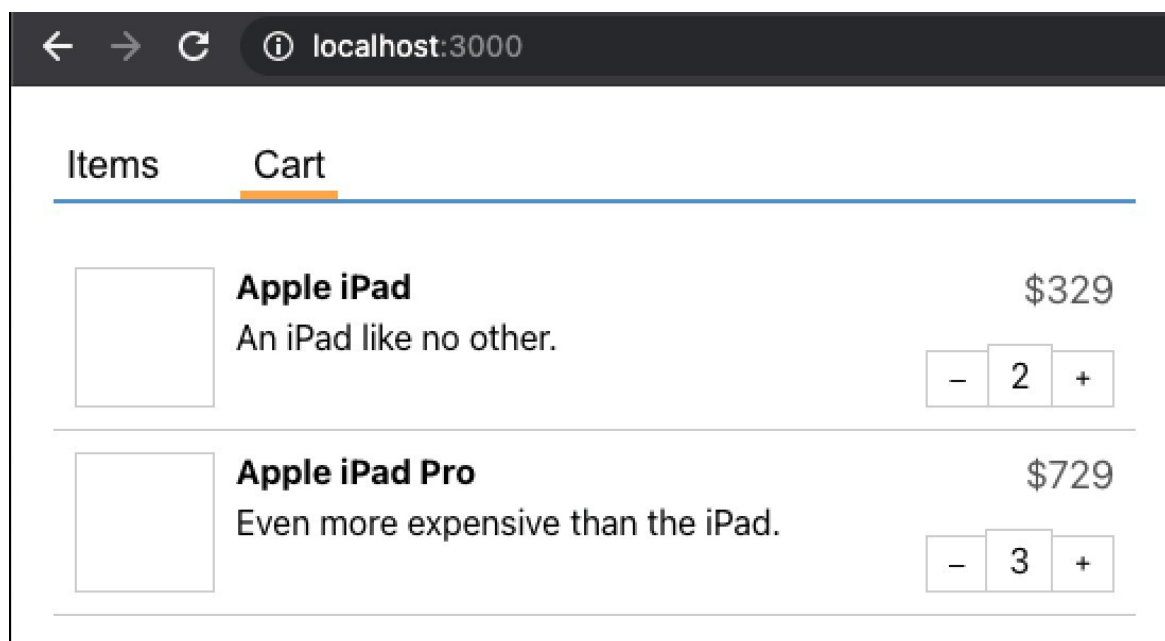
In this lesson we implement the **CSS styling** for our **CartPage** Component. The image below, from our design, shows the result of our **CartPage** Component styling.



CartPage Component – CSS Styling

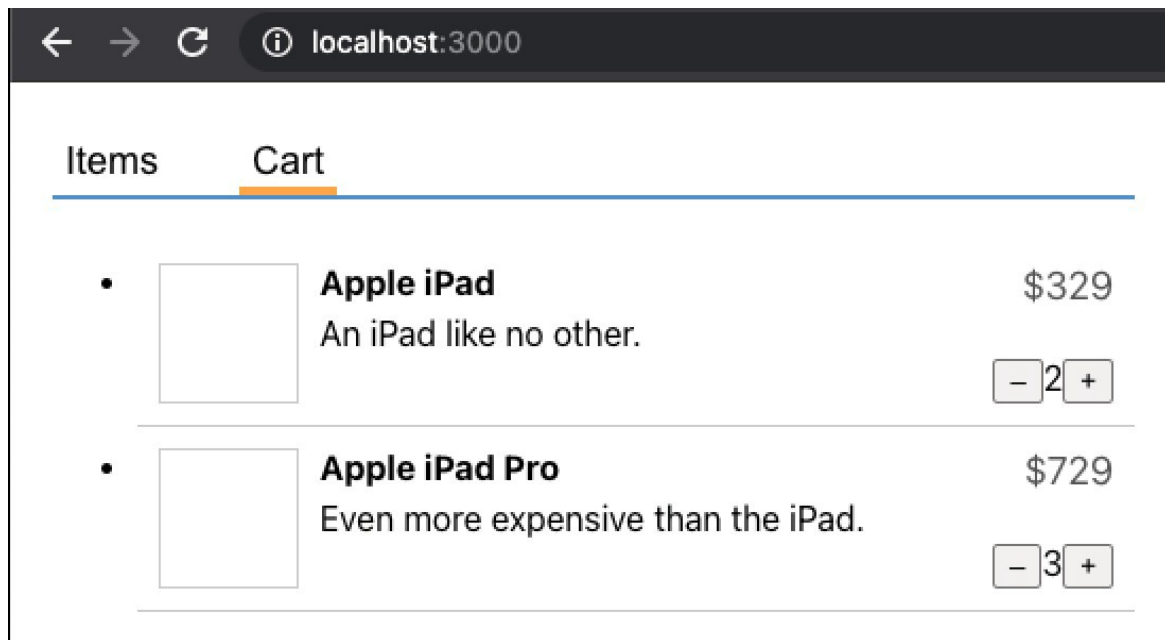
It's now time to provide the **CSS styling** to all the **CSS classes** we used in the **CartPage** Component. The two images that follow show the difference CSS makes to our webpage. The first image is before styling the **CartPage** Component and the second image is the same content with CSS styling applied.



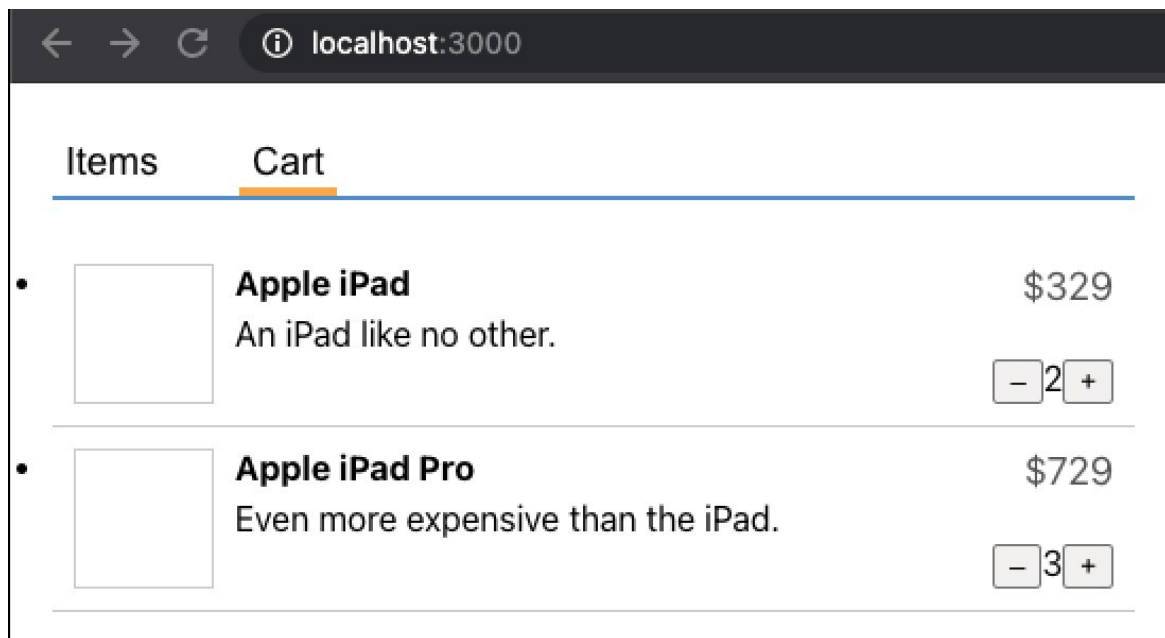


Open the file name **CartPage.css** (previously **created** and **imported** in **CartPage.js**). You can type in the CSS styles, **copy** and **paste** from the **CartPage.css** in the course downloads or from the code block below. What follows is the **complete** CSS for the **CartPage** Component. The next section illustrates step by step how our **CSS** transforms the **CartPage** Component between the before and after images shown above. Each **CSS** rule **targets** a **CSS** class or **element** from our **CartPage** Component.

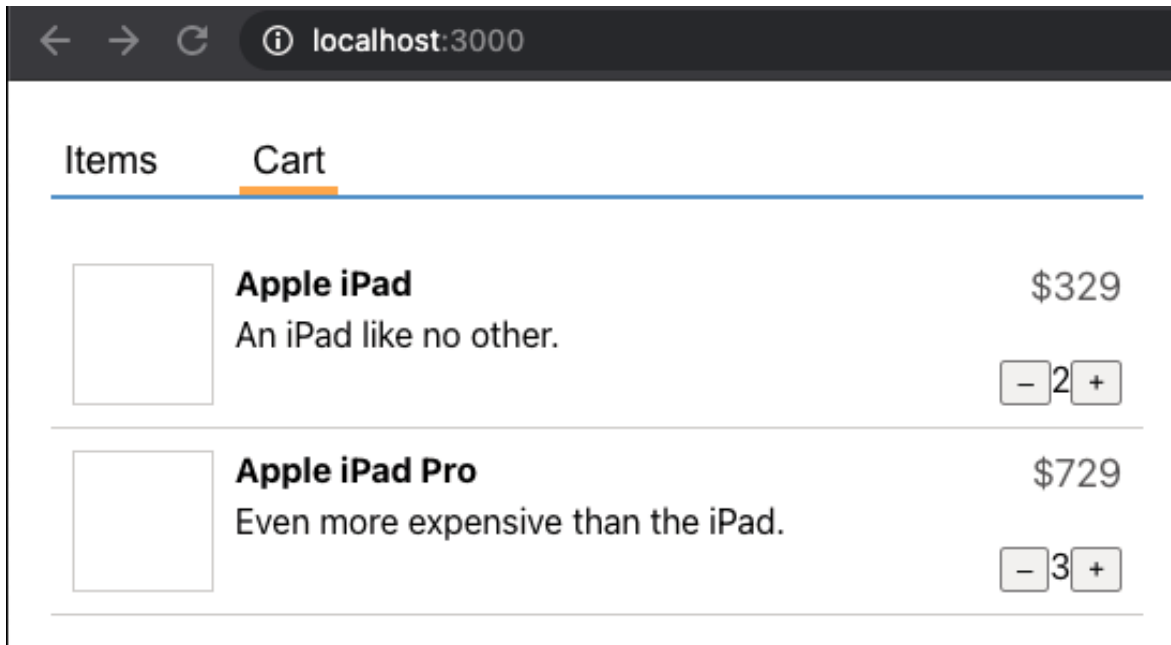
```
.CartPage-items {
  margin: 0;
  padding: 0;
}
.CartPage-items li {
  list-style: none;
}
.CartItem-count {
  padding: 5px 10px;
  border: 1px solid #ccc;
}
.CartItem-addOne,
.CartItem-removeOne {
  padding: 5px 10px;
  border: 1px solid #ccc;
  background: #fff;
}
.CartItem-addOne {
  border-left: none;
}
.CartItem-removeOne {
  border-right: none;
}
```



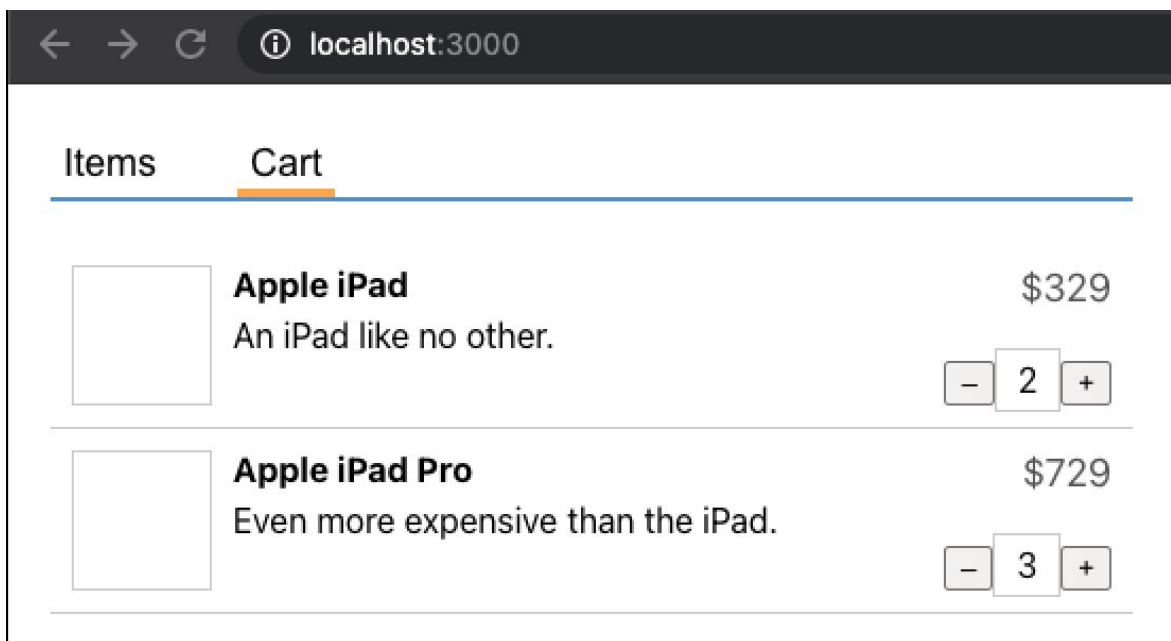
```
.CartPage-items {  
  margin: 0;  
  padding: 0;  
}
```



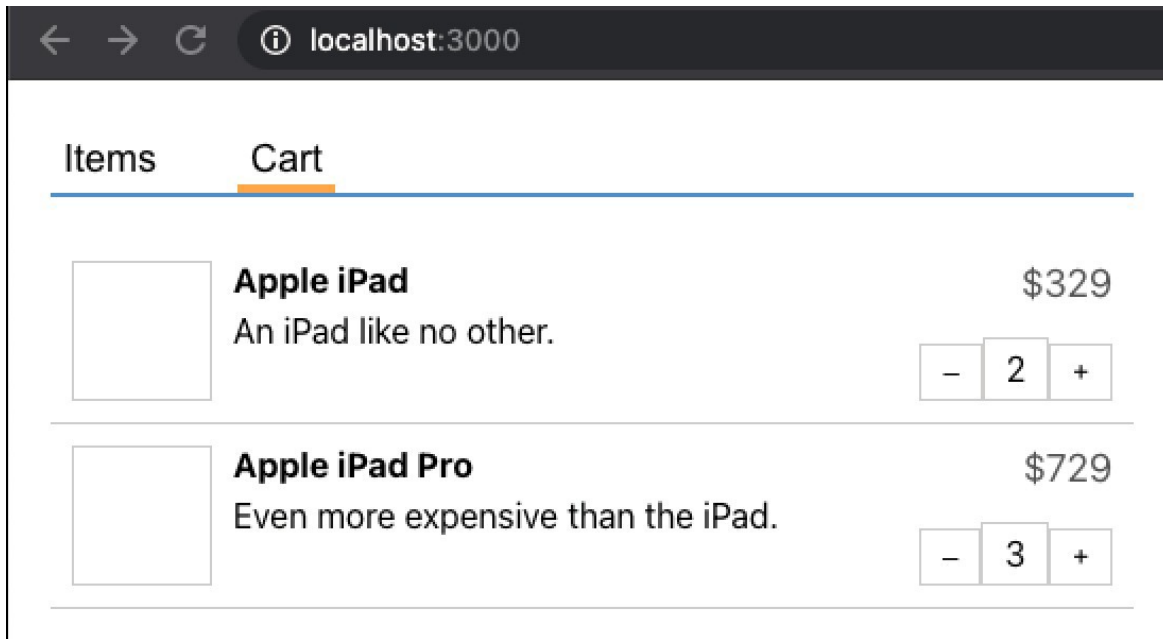
```
.CartPage-items li {  
  list-style: none;  
}
```



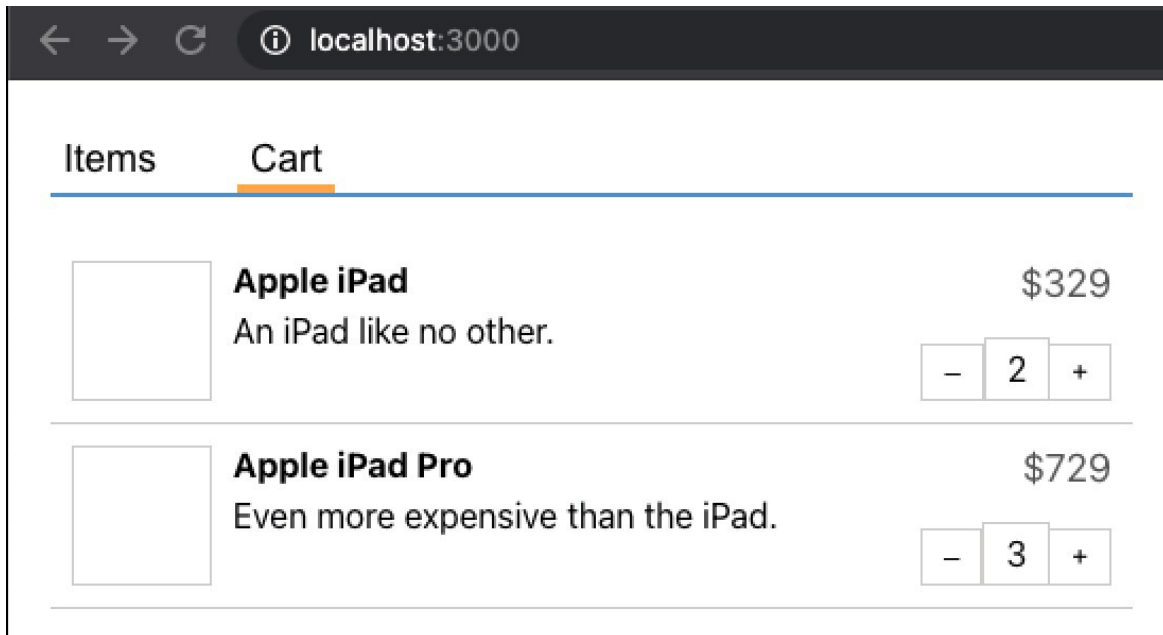
```
.CartItem-count {  
  padding: 5px 10px;  
  border: 1px solid #ccc;  
}
```



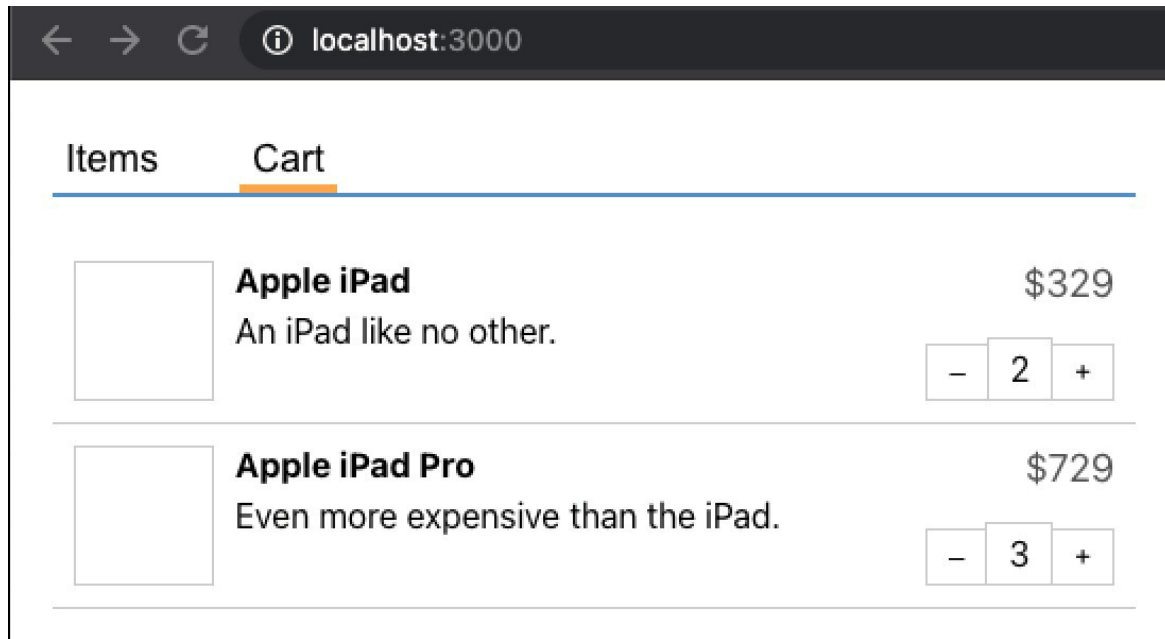
```
.CartItem-addOne,  
.CartItem-removeOne {  
  padding: 5px 10px;  
  border: 1px solid #ccc;  
  background: #fff;  
}
```



```
.CartItem-addOne {  
  border-left: none;  
}
```



```
.CartItem-removeOne {  
  border-right: none;  
}
```



As you can see, our app looks better with a little CSS styling. Congratulations on completing this Shopping Site with **React!!!**